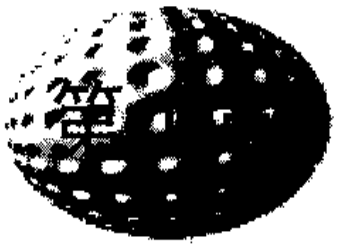


基础知识

目前，计算机技术已渗透到人类生活的方方面面。计算机有这么大的本领,是由于人们利用计算机记忆能力强、计算速度快的固有特点，为它事先编制好了处理各种问题的程序。计算机完全是照程序中的指令，按部就班地执行的。那么，什么是程序？如何编制程序？这正是我们这门课程要学习的内容。

本章简要介绍程序设计的相关概念、Pascal语言的基础知识和TURBO Pascal上机方法。

1.1 程序和菜谱

下面举一个日常生活中的例子，来说明什么是程序.

小明要学做“西红柿炒鸡蛋”这道菜，爸爸给了她一张菜谱：

菜名：西红柿炒鸡蛋.

配料：西红柿4个，小葱3根，鸡蛋2个，盐1匙.

操作步骤:

1. 将洗净的西红柿切成小块;
2. 将洗净的小葱切成小段，装入碗中;
3. 将鸡蛋打入盛小葱的碗中，搅匀;
4. 首先在锅中放一勺油，然后点燃煤气灶，并将火力调至最大;
5. 等待约30秒;
6. 将鸡蛋倒入油锅，反复翻炒，直至凝固;
7. 把火调至最小;

8. 将炒好的鸡蛋装入碗中；
9. 锅中放一勺油，再将火调至最大；
10. 等待约30秒；
11. 西红柿入油锅，翻炒数下；
12. 鸡蛋入锅，放一匙盐，和西红柿一起翻炒数下；
13. 关掉煤气灶，装盘。

这张菜谱实际上就是一个程序：程序处理的对象是配料清单上所列的各种原料；程序设计者是小明的爸爸；程序执行者是小明；所用的程序设计语言是汉语。

我们要学习的是如何设计计算机程序。计算机程序与菜谱之类的普通程序非常相似，但也有不同之处。在开始学习如何设计计算机程序之前，我们先将计算机程序和菜谱之类的普通程序作一番对比，看看两者之间有哪些异同之处：

① 程序要用语言来描述。炒菜程序的执行者是小明，所以炒菜程序要用她看得懂的汉语来描述。而计算机程序的执行者是计算机，所以计算机程序要用计算机语言来描述。

② 程序中的每一个步骤必须是明确的、可执行的。小明第一次学习炒菜，爸爸给她的菜谱每一步都写得很清楚，例如，没有使用一般供成年人阅读的菜谱书籍中常见的“放少许盐”、“油热至八成”等儿童不易把握的术语。计算机是没有思维能力的，为计算机写的程序每一步更要仔细精确。像炒菜程序中“等待约30秒”这样的指令，一个小学生能理解，计算机却无法执行，我们应该明确告诉计算机确切的等待时间。

③ 程序都有需要处理的对象。一个炒菜程序要处理的对象是各种配料；一个计算机程序要处理的对象是各种数据 。

④ 在外界因素不变的情况下，同一个程序处理相同的对象时，将产生相同的结果。对于炒菜程序来说，只要小明严格按照上面的菜谱执行，每次都能得到口味大致相同的“西红柿炒鸡蛋”。对于某个计算机程序来说，只要输入数据不变，每次执行这个程序，将得到完全相同的处理结果。

通过以上对比，可以发现，编写计算机程序和编写炒菜程序原理是一样的。要设计一个计算机程序，首先，目的要明确，程序要处理什么对象？希望得到什么结果？在编写程序前就必须心中有数；其次，步骤要清楚，程序先做什么后做什么？每一步怎么做？在程序中必须写清楚；最后，语言要合适，要用程序的执行者“看得懂”的语言来编写程序。

1.2 计算机语言

计算机语言有低级语言和高级语言之分.

低级语言又称机器语言，它只有0和1两种符号。通过一长串0和1的不同组合来表示不同的指令(称为机器指令)。计算机硬件系统只能执行用机器语言编写的程序。

用机器语言来编写程序是一件非常麻烦的事情，而且编写出来的程序不具备通用性，因为不同类型计算机所用的机器语言是不相同的。

高级语言又称算法语言，它具有两大特点：

① 与自然语言和数学语言非常接近，一般人比较容易掌握；

② 与具体计算机无关，即用它编写的程序可以在任何一种计算机上运行（必要时只需做一些很小的改动）。

平常人们所说的“计算机程序设计语言”一般都是指高级语言.

高级语言有很多种，如Pascal语言、Basic语言、C语言等，都是被广泛使用的高级语言.

1.3 编译程序

计算机硬件系统只懂机器语言，只能执行用机器语言编写的程序（称为目标程序），不能直接执行用高级语言编写的程序（称为源程序）。

为了使计算机能够执行高级语言程序，我们需要在计算机系统中安装一个翻译软件，这个翻译软件称为编译程序。编译程序的作用是把源程序翻译成目标程序.

编译程序是由专业人员设计的，作为一种产品可在电脑商店里买到.

用高级语言编写的源程序在计算机上执行，要分两个阶段：

① 编译阶段 把源程序输入到计算机中，由编译程序自动地将它翻译成计算机硬件能够执行的目标程序，如图1.1所示.

② 运行阶段 执行目标程序，输入要处理的数据，获得处理结果，如图1.2所示.

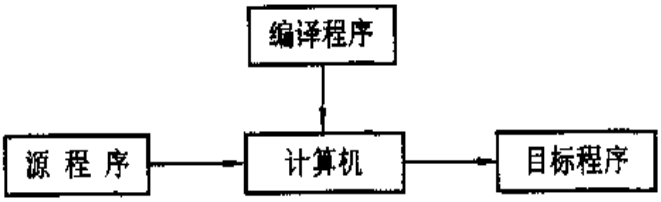


图1.1　编译阶段

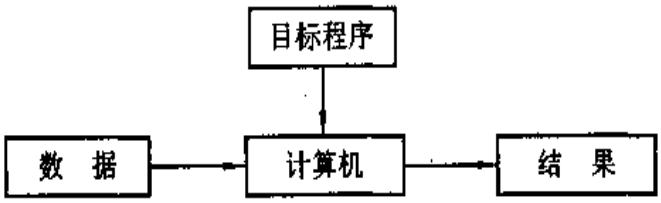


图1.2　运行阶段

1.4　Pascal语言

Pascal语言是一种高级语言，是由瑞士科学家沃思教授于1968年设计的。

Pascal语言有许多种版本，如MS Pascal、UCSD Pascal、TURBO Pascal等。各种版本的Pascal语言有些差异，但基本内容相同，都是在沃思教授当年设计的Pascal语言基础上发展起来的。

TURBO Pascal是目前使用最广的程序设计语言之一，它也有许多种版本，最新版本是Pascal 7.0。

本书介绍如何用TURBO Pascal语言编写程序。TURBO Pascal语言功能很强，内容非常丰富，本书只介绍其中最基本的部分。

1.5　Pascal程序

用Pascal语言编写的程序称为Pascal程序。在学习用Pascal语言编写程序之前，让我们先来看一个简单的Pascal程序。这个程序的功能是：从键盘输入一个整数，作为圆的半径，计算出这个圆的周长和面积，并将计算结果显示在屏幕上。

```
program example;{程序首部}
 const pi=3.14;  {圆周率}
 var r:integer;
     c,s:real;
 begin
  write( 'r=' );
  read(r);
  c:=2*pi*r;
  s:=pi*r*r;
  writeln('c=',c);
  writeln('s=',s)
 end.
```

说明部分 执行部分 程序体

Pascal程序一般包括3个部分：程序首部、说明部分、执行部分．其中，说明部分和执行部分合在一起又称为程序体．

1.5.1 程序首部

程序首部包括程序标志和程序名．

1. 程序标志

英文单词“program”是“程序”的意思，Pascal语言把它作为程序开始的标志。

2. 程序名

程序的名字由程序设计者自己命名．虽然，给程序取什么名字不会影响到程序的执行结果，但是，为了提高程序的可读性，应尽可能取有意义的名字．另外，给程序取名字时，还要遵循1.6.3节所述用户标识符的选取规则．

程序的名字相当于炒菜程序中的菜名．

在TURBO Pascal程序中，程序首部是可以省略的．

1.5.2 说明部分

程序的说明部分用来描述程序中用到的数据．在上述程序实例中，程序的说明部分说明了4个数据：圆周率π，圆半径r，圆周长c和圆面积s．π

是个常数，作者用标识符pi作为它的名字（π不是英文字母，不能用做名字），取近似值3.14。这个程序只计算以某个整数值为半径的圆的周长和面积，整数值从键盘输入。所以，将半径r说明为integer(整数)类型的变量，将周长c和面积s说明为real(实数)类型的变量。

关于常量、变量和数据类型的概念将在1.7节和1.8节中详细介绍。

Pascal程序的说明部分相当于炒菜程序中的配料清单。

除了那些特别简单的程序以外，一般的Pascal程序都有说明部分。

1.5.3　执行部分

程序的执行部分以“begin”开始，“end”结束，“begin”和“end”之间是描述对数据如何处理的各种语句。各语句之间要用分号“；”分隔。

在程序首部、说明部分和执行部分之间要用分号“;”分隔，执行部分的最后要写上一个句号“.”表示整个程序的结束。

程序的执行部分相当于炒菜程序中的操作步骤。

综合以上所述，一个Pascal程序具有如下形式：

```
program 程序名;
说明部分;
begin
  语句组
end.
```

或者

```
说明部分;
begin
  语句组
end.
```

Pascal程序的书写方法比较灵活：允许在一行中写几个语句，也允许一个语句分几行写。例如，程序example也可以写成：

```
program example; const pi=3.14; var r:integer; c,s:real; begin write ('r='); read(r); c:=2*pi*r; s:=pi*r*r; writeln('c=',c); writeln('s=',s) end.
```

不过这样书写的程序可读性较差，应该尽量避免。

我们编写的程序，除了要供计算机执行外，还要用于同行之间的交流。因此，应该使程序结构清晰、容易阅读。这是一个“好程序”的重要标志。

为了提高程序的可读性，可以在程序的适当位置插入注释。Pascal程序中的注释要用一对花括号“{”和“}”括起来，内容可根据需要书写。注释是给自己或他人看的，Pascal语言的翻译软件不翻译注释内容。

1.6 Pascal符号

Pascal程序要用Pascal符号来表示．Pascal符号包括特殊符号、预定义标识符和用户标识符3大类。

1.6.1 特殊符号

特殊符号是指在程序中具有特殊含义的符号，共有3种：

1. 单个特殊字符

+ - * / = > < ( ) [ ] { } . ; : ' ^

2. 一对特殊字符

:= >= <= <> ..

一对特殊字符在书写时不可以分开．例如，不可以把“:=”写成“: =”．

3. 保留字

保留字是一批英文单词，它们在程序中有特殊的含义，不可挪作他用．例如，不可以用“begin”做程序名，因为“begin”是保留字。

TURBO Pascal语言的保留字有47个．作为程序设计的基础教程，本书将介绍下列33个保留字的用法：

and	array	begin	case	const
div	do	downto	else	end
file	for	function	if	in
mod	nil	not	of	or
procedure		program	record	repeat
set	string	then	to	type
until	var	while	with	

下列14个保留字的用法涉及到较复杂的问题，本书不介绍：

absolute	external	goto	implementation	inline
interface	interrupt	label	packed	shl
shr	unit	uses	xor	

1.6.2 预定义标识符

预定义标识符是一批英文单词（或英文单词的缩写），在程序中也有特殊的含义。和保留字不同的是，允许程序设计者重新定义这些标识符，将它们作为一般的用户标识符来使用。但是，对于初学者来说，最好不要这样做。

TURBO Pascal语言的预定义标识符有200多个，本书中只出现了其中的49个：

abs	assign	blockread	blockwrite	boolean
char	chr	close	concat	copy
cos	delete	dispose	eof	eoln
exp	false	filesize	forward	insert
integer	length	ln	new	ord
pos	pred	random	randomize	read
readln	real	reset	rewrite	round
seek	sin	sizeof	sqr	sqrt
str	succ	text	true	trunc
upcase	val	write	writeln	

1.6.3 用户标识符

用户标识符是由程序设计者根据需要自己定义的，用来作为常量、变量、类型、函数、过程、程序等的名字。

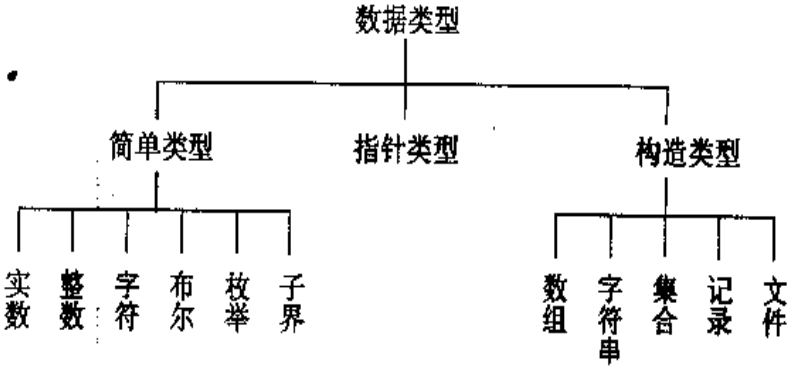
用户标识符必须以英文字母开头，后面可以跟若干个字母或数字。标识符的长度可以是任意的，但只有前63个字符有效。

出现在标识符（包括保留字、预定义标识符和用户标识符）中的字母是不分大小写的。因此，begin和BEGIN是同一个保留字；READ和Read是同一个预定义标识符；example和EXaMpLe是同一个用户标识符。

1.7 数据类型

计算机程序处理的对象是数据，而数据的一个重要特征是它的类型。对某个数据能进行哪些运算取决于该数据的类型。

Pascal语言的数据类型极为丰富，本书将介绍其中的12种。这12种数据类型可分为以下3大类：



简单类型也称基本类型。简单类型的数据是不可分解的。

构造类型也称复杂类型。构造类型的数据是由其它类型的数据按一定规则构成的。

指针类型实际上也是一种简单类型，但它是用来构造动态数据的。

除实数以外，其它5种简单类型都是有序类型。所谓有序类型是指每个这种类型的数据值都对应一个序号。假设x为有序类型的数据，则它所对应的序号可用标准函数ord(x)求得。例如：

ord (FALSE)=0　　ord (TRUE)=1

ord ('A')=65　　ord ('B')=66

ord (20)=20　　ord (-1)=-1

注意：实数类型不是有序类型。如果x不是有序类型的数据，则不可以使用标准函数ord(x)。

简单类型中的实数、整数、字符、布尔都是标准类型，有预定义类型标识符：

实数类型	整数类型	字符类型	布尔类型
real	integer	char	boolean

在程序中可以直接用这些类型标识符来说明有关的变量。在第5章之

前，我们只使用这 4 种标准类型。

1.8 常量和变量

在程序中，数据或者以常量的形式出现，或者以变量的形式出现。常量和变量都是有类型的。本章只介绍整型、实型、字符型数据，其它类型的数据在后续章节中介绍。

1.8.1 常量

在程序运行过程中，其值保持不变的量称为常量。

不同类型的常量在程序中有不同的书写形式，可以根据书写形式判断常量的类型。

1. 整型常量

整型常量包括正整数、负整数和零。

在Pascal程序中，整型常量的书写方法与平常使用的十进制整数书写方法相同。例如，

6　–13　+26　0

都是书写正确的整型常量。但下列写法都不正确：

6.0　　(应写成6)

6,543　(应写成6543)

0.0　　(应写成0)

注意：书写整型常量时，数字之间不可以有小数点或逗号。

计算机程序所能处理的数据都有一个范围，这个范围的大小与计算机硬件有关。对于integer 类型的数据来说，最大值是32767，最小值是–32768。如果整数超出了这个范围，则发生错误。

2. 实型常量

实型常量包括正实数、负实数和实数零。

在Pascal程序中，实型常量有两种表示形式：

(1) 十进制形式

十进制形式就是平常使用的小数形式表示方法。例如，

–1234.5　6.0　0.78　+9.10

都是书写正确的实型常量．但下列写法都不正确：

0. (应写成0.0或0)

.9 (应写成0.9)

–.123 (应写成–0.123)

注意：实型常量用十进制形式表示时，一定要有小数点，而且小数点前后一定要有数字。

(2) 指数形式

指数形式就是在十进制整数或十进制实数的后面接字母E（或e），在这个字母的后面再接一个十进制整数．例如，

6.25E8 (等于6.25乘以10的8次方)

–51e–8 (等于–51乘以10的负8次方)

123E4 (等于123乘以10的4次方)

都是书写正确的指数形式的实型常量．但下列写法都不正确：

.12E3 (应写成 0.12E3)

4.E–5 (应写成 4.0E–5)

6.7E8.0 (应写成 6.7E8)

E9 (应写成 1E9)

注意：实型常量用指数形式表示时，字母E（或e）的前面要有整数或十进制实数；E（或e）的后面要有整数。

real类型数据的表示范围在绝对值2.9E–39至1.7E38之间，如图1.3所示。

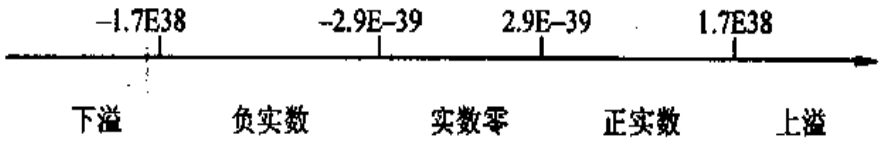


图1.3 real类型数据的表示范围

real类型数据的绝对值如果大于1.7E38，则出错；如果小于2.9E–39，则视为零（称为实数零）。

real类型数据的表示范围虽然很大，但有效位数只有11位．例如，实数

0.12345678901234

和

0.12345678901239

被计算机认为是两个相等的实数，都是 1.2345678901E–01,而无法区分．

3. 字符常量

字符常量是一个用单引号括起来的字符。例如，'9'，'*'，'y'都是字符常量。字母作为字符常量使用时，大小写是有区别的，例如，'A'和'a'是两个不同的字符常量。

如果想表示单引号字符常量，则应该写成''''，即连写4个单引号。

允许在Pascal程序中用作字符常量的字符共有95个，见附录B。

4. 常量说明

常量说明就是给常量取个名字，用名字来表示某个常量。常量说明要放在程序的说明部分，用保留字const作标志。一般形式为

```
const  常量名 = 常量
```

其中，常量名是一个用户标识符。前面的example程序实例中第2行

```
const pi=3.14;
```

就是一个常量说明，标识符pi经过这样说明之后，就有值3.14。

在程序中给某些常量取名字，使用经过命名的常量，其好处是使得程序容易修改。在example程序的执行部分，有两个地方用到了圆周率π的近似值3.14。程序设计者没有直接用常量值3.14，而是用了经过说明的常量名pi，这样，如果要修改程序，取π的近似值为3.14159，使计算结果更精确，则只需修改常量说明

```
const  pi=3.14159;
```

而不必去修改程序的执行部分。

给常量取什么样的名字，没有特别的规定，只要符合用户标识符命名规则即可。但是一般来说，最好取有意义的名字。例如，将π的近似值这个常量命名为pi就比较好，因为pi的读音和π的读音相似，让人一看便可知道它代表的是π。

1.8.2　变量

在程序运行过程中，其值可以改变的量称为变量。

每个变量必须用一个合适的用户标识符来命名，这个标识符称为变量名。

example程序实例中表示圆的半径、周长和面积的3个用户标识符r,c和s就是3个变量名。

变量在使用之前都必须先进行说明。变量说明用保留字var 作标志，放在程序的说明部分，其一般形式为

var 变量表:类型

其中，变量表是一系列用逗号分隔的变量的名字，这些变量具有相同的类型。例如，在程序实例example 中有变量说明

```
var
  r:integer;
  c,s:real;
```

它说明r是整型变量，c和s是实型变量。

为什么要对变量进行说明呢?

在计算机中，不同类型的数据所占用的存储空间的大小是不同的。例如，integer类型的数据占用2个字节的存储空间；real类型的数据占用6个字节的存储空间；等等。有了变量说明，计算机就可以给每个变量分配相应的存储空间。

另外，不同类型的数据所能进行的运算种类和取值范围也不一样。例如，real类型和integer类型的数据可以进行加、减、乘、除等算术运算，而char类型的数据则不可以；real类型数据的最大值是1.7E38；integer类型数据的最大值是32767。如果你在源程序中错误地使用了某个变量，例如，企图对char类型的数据进行加、减、乘、除运算，或者，企图把一个大于32767的数存入integer类型的变量中，计算机在编译该源程序时，可以根据所写的变量说明发现并指出这些错误。

1.9 算术表达式

算术表达式是由一些整型或实型的数据、算术运算符和圆括号组成的计算式，计算结果是一个整数或实数。

根据计算结果类型的不同，算术表达式分整型表达式和实型表达式两种。

在整型表达式中，作为运算对象的数据必须是整数类型的。可以使用的算术运算符有5个：

+	-	*	div	mod
(加)	(减)	(乘)	(整除)	(取余)

注意：div和mod这两个运算符是保留字。

若x和y都是整型数据，则算术表达式

x div y

和

x mod y

的值分别为x被y除所得整数商和余数。例如，

15 div 3=5

16 div 3=5

1 mod 3=1

5 mod 3=2

在实型表达式中，作为运算对象的数据可以是实数类型的也可以是整数类型的。可以使用的算术运算符只有4个：

+ – * /

(加) (减) (乘) (除)

读者可能已注意到，整型表达式中的加、减、乘运算和实型表达式中的加、减、乘运算使用同一组符号(+, –, *)．对于算术表达式

a+b

a–b

a*b

来说，只有当a和b都是整型数据时，上述3个表达式的计算结果才是整型的；否则，只要a和b中至少有一个是实型数据，则计算结果就是实型的．例如，

5*2=10 （两个运算对象都是整型数据，所以计算结果是整型的）

5*2.0=10.0（有一个运算对象是实型数据，所以计算结果是实型的）

但是，对于算术表达式

a/b

来说，无论a和b是整型数据还是实型数据，计算结果总是实型的．例如，

5/2=2.5

5/2.0=2.5

1.10 标准函数

在编写程序的时候，经常会遇到一些复杂的计算．例如，计算三角函数值，计算平方根，等等．为了方便用户编写程序，Pascal语言系统的设计

者已预先为我们写好了完成这些计算的程序段，这种具有特定计算功能的程序段称为标准函数。例如，求x的正弦函数值可用标准函数sin(x)；求x的平方根可用标准函数sqrt(x)。

每个标准函数都有自己的名字。标准函数名属于预定义标识符，有特定的含义，在程序中尽量不要挪作它用。

标准函数处理的对象称为参数。例如，sin(x)和sqrt(x)的参数是x。参数的名字并不重要，重要的是参数的类型。每个标准函数可以计算哪些类型的参数，系统有严格的规定。

TURBO Pascal系统总共提供了100多个标准函数。本书将通过具体实例介绍其中23个函数的用法。表1.1中列出了这些标准函数的名字、功能和应用实例在本书中的页码。

表1.1 常用标准函数

函数名	功 能	索引	函数名	功 能	索引
abs	取绝对值	56	pos	求子串在字符串中的位置	122
chr	根据序号求字符	151	pred	求有序类型数据的前趋值	61
concat	联接字符串	121	random	返回一个随机数	110
copy	求子串	121	round	四舍五入	46
cos	求余弦函数值	46	sin	求正弦函数值	46
eof	测试是否遇文件结束符	82	sizeof	返回数据所占内存字节数	214
eoln	测试是否遇行结束符	219	sqr	求平方值	63
exp	计算e^x值	46	sqrt	求平方根值	30
filesize	返回文件长度	210	succ	求有序类型数据的后继值	61
length	返回字符串长度	122	trunc	截取实数的整数部分	83
ln	计算自然对数值	46	upcase	小写字母转换成大写字母	173
ord	求有序类型数据的序号	9			

1.11 TURBO Pascal 的集成环境

TURBO Pascal系统有一个称为集成环境（IDE）的应用软件，它集编辑、编译、调试、运行于一体，功能强大、使用方便。本节将简要介绍在这个集成环境下输入、运行Pascal程序的方法，其中只涉及IDE的一小部分

功能。

1.11.1　IDE的进入和退出

在操作系统的命令状态下，键入

TURBO

系统就进入了TURBO Pascal的集成环境，其主屏幕如图1.4所示。

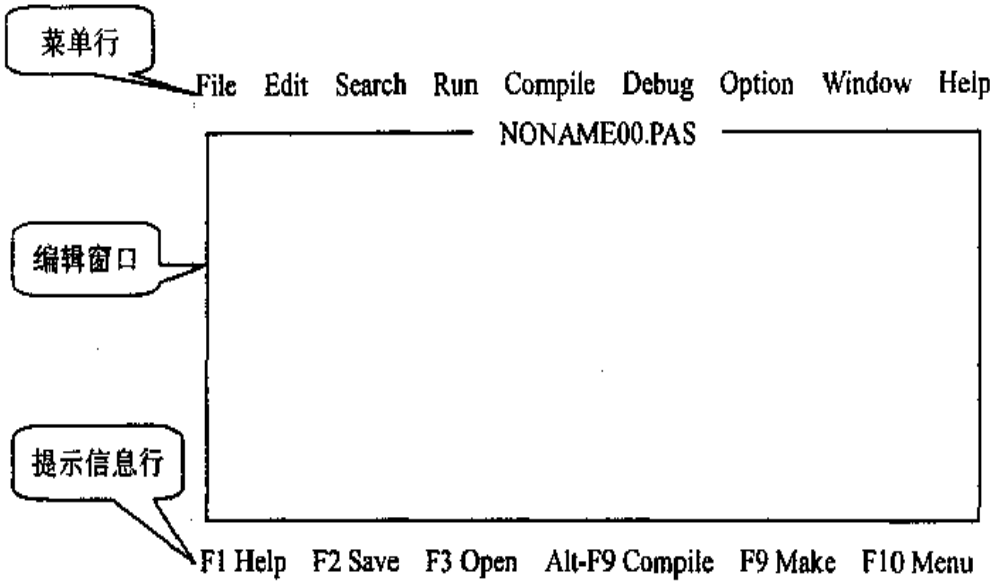


图1.4　IDE的主屏幕

主屏幕分为3个区域：上面的菜单行、中间的编辑窗口和下面的提示信息行。

刚进入IDE时，光标位于编辑窗口，系统处于编辑状态。此时，如果输入源程序，输入的内容将在编辑窗口中显示。

按F10键，可将光标移到菜单行。当光标位于菜单行时，按←或→键，可使光标在菜单行中左右移动。当光标移到你所要的选项处时，按回车键，就可以选中该选项。

在菜单行中，自左至右共有9个选项：

File	文件处理
Edit	编辑
Search	查找
Run	运行
Compile	编译
Debug	调试处理

Option 选择项处理
Window 窗口处理
Help 帮助

当选中了某个选项时，在该选项的下面会出现一个子菜单。此时，按↑或↓键，可使光标在子菜单中上下移动；按回车键，将执行光标所在处的命令。

如果想让光标从菜单行返回到编辑窗口，可按Esc键。

如果想退出IDE，返回到操作系统的命令输入状态，可按Alt-X键(Alt-X表示同时按Alt和X键，以下类推)。此时，如果编辑窗口中的内容没有存盘，系统会提示你是否希望存盘。

1.11.2 常用编辑命令

源程序的输入是在编辑状态下进行的。系统提供了几十条编辑命令，对于初学者来说，只要掌握其中一些常用的命令就可以了。

1. 移动光标

按↑、↓、←、→键，可使光标在编辑窗口上、下、左、右移动。

2. 翻页

当程序较长，超出窗口高度时，如果想看上一页或下一页的内容，可按PgUp或PgDn键。

3. 删除一个字符

如果在输入的过程中，发现某个字符输错了，可先将光标移到该字符所在位置，然后按Del键删除之。此时，位于该字符右边的所有字符会自动左移一个位置。

4. 删除一行

如果要删除某一行，可先将光标移到该行任何位置处，然后按Ctrl-Y键。

5. 块操作

利用块操作可以快速地删除某一段程序(称为块)，或将该程序段拷贝到光标所在任何位置。步骤如下：

① 做块首标记（将光标移到块的前面，按Ctrl-K-B键）；

② 做块尾标记（将光标移到块的后面，按Ctrl-K-K键）；

③ 如果想删除这一块，则按Ctrl-K-Y键；

④ 如果想把这一块拷贝到程序某一位置，则先移动光标至该位置，然后按Ctrl-K-C键。

1.11.3 程序的编译和运行

在输入一个源程序之前，首先要给这个程序文件取个名字。按F3键，屏幕中央会出现一个对话框，在此框中键入即将输入的源程序文件名（例如ch1），然后按回车键，则编辑窗口顶部会出现源程序文件名ch1.pas。光标落在编辑窗口的左上角。接着，就可以输入源程序了。

在输入一个源程序之后，按Alt-F9键，系统对源程序进行编译，这期间屏幕上会出现一些编译信息。

如果源程序中有语法错误(例如，保留字拼错，两个语句间少分号，等等)，系统会将程序中所有语法错误的提示信息按出现的先后顺序显示在屏幕上。按任意一个键，系统又回到编辑状态，光标落在出现第一个语法错误处。此时，你要做的事情就是改正这些错误。

附录A列出了常见语法错误的错误编号和错误原因。

注意：编译程序指出的出错位置是一个大致位置，不一定准确。有时，编译程序认为有错的地方，实际上是完全正确的，错误出在前一二行中。有时，前面一个语法错误，会导致后面许多正确的程序行也被认为有错。

在将源程序中的语法错误全部排除之前，系统不生成目标程序。

如果源程序中没有语法错误，或语法错误已被排除，编译结束后屏幕上就会出现编译成功的提示信息

Compile successful:Press any key

此时，目标程序已生成，你可以先按任意一个键，回到编辑状态，然后按Ctrl-F9键，执行这个目标程序。

在执行目标程序过程中，可能会出现运行错误(例如，出现除数为零的情况)。此时，屏幕上会出现提示信息

Run time error <错误编号> at < 内存地址 >

并停止执行程序。

目标程序执行结束后，系统自动返回编辑状态。此时，如果你想查看一下程序的输出结果，可按Alt-F5键。看过之后，按任意一个键又可回到编辑状态。

1.11.4 存储程序

在编辑状态下输入的源程序是暂存在内存中的，一旦关机或退出IDE，所输入的内容就再也找不到了。为了防止在输入过程中意外断电而导致输入的内容全部丢失，在输入过程中，可以每隔一段时间就执行一次存盘命令，把已输入的内容永久保存到磁盘上。存盘命令非常简单，按一下F2键即可。源程序文件名与编辑窗口顶部显示的文件名相同。

习 题 一

1.1 填空题。

(1) 只有“0”和“1”两种符号的语言称为_____语言。

(2) Pascal语言是一种_____语言。

(3) 编译程序的作用是_____。

(4) Pascal程序的首部包括_____和_____。

(5) Pascal程序的执行部分以_____开始，_____结束。

(6) Pascal语言的符号包括_____、_____和_____三大类。

(7) 计算机程序处理的对象是_____。

(8) 常量说明以_____作标志；变量说明以_____作标志。

(9) integer类型数据的表示范围是_____。

(10) 实型常量有十进制形式和_____两种表示方法。

(11) 源程序在计算机上执行要经过_____和_____两个阶段。

(12) IDE的主屏幕分为_____、_____和_____3个区域。

1.2 判断题。

(1) 低级语言比高级语言更容易学。

(2) 程序说明部分的作用是说明程序要做什么事情。

(3) 没有说明部分的程序一定是错误的程序。

(4) 程序首部是可以省略的。

(5) 语句和语句之间要用逗号分隔。

(6) 标准函数名属于保留字。

(7) Pascal程序的一行可以写多个语句。

(8) 有序类型是指每个这种类型的数据值都对应一个序号。

(9) real类型属于有序类型。

(10) 实数零是指那些绝对值小于2.9×10^{-39}的数。

(11) 计算机硬件系统不能执行Pascal程序。

(12) 源程序文件名要和程序名相同。

1.3 指出下列常量说明有哪些错误：

```
const
 min=10000;
 max= -1000;
 end='#';
 five=2+3;
 date=2000.01.01;
 small=max;
 data=100,000;
 smallest=1e-10.0;
 largest=12e3;
 -b=max;
```

1.4 指出下列变量说明有哪些错误：

```
var
 a,b:integer;
 B,C:integer;
 10*x:real;
 d:character;
 100:integer;
 const:real;
 sun:char;
 beginend:char;
```

1.5 写出下列算式对应的Pascal表达式：

(1) $2a_1+3a_2-4a_3$

(2) $2\pi r^2$

(3) $\dfrac{x-1}{x+1}$

(4) $\sqrt{a^2+b^2}$

(5) $(a+b)\div2$

1.6 写出下列Pascal表达式的值：

(1) 5/2+2

(2) 5 div 2*2

(3) 50 div 4 mod 5

(4) 15 div(15 mod 17)

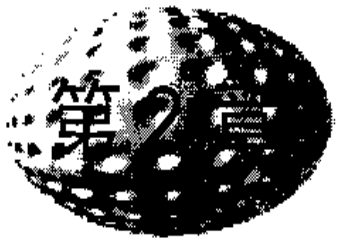
1.7 输入并运行1.5节中给出的程序,检查运行结果:

(1) 如果将程序名example改成error,对程序的运行结果有什么影响?

(2) 如果将第2行中的注释{圆周率}写成{园周率}对程序的运行结果有什么影响?

(3) 如果将第3行var r:integer中的r写成了R,会出现什么错误?

(4) 如果将第2行const pi=3.14; 写成const pi=314; 或将第8行c:=2*pi*r; 写成 c:=2*pi*r*r; 计算机系统会指出这些错误吗?



程序的顺序结构

一个求解实际问题的Pascal程序可能是非常复杂的，包含有成百上千条语句。然而，不管程序多么复杂，它总是由一些基本语句构成的，并且具有一定的结构。本章将介绍几个最重要的基本语句，以及由这些语句所构成的最简单的程序—顺序结构程序。

2.1 赋值语句

这是一个在程序中用得最多的基本语句。计算机对数据进行处理的工作主要是由赋值语句完成的。

赋值语句的格式为

```
名字:=表达式
```

其中，“:=”是赋值号，它由冒号和等号组合而成，书写时要注意在冒号和等号之间不能夹入空格，也不可以写成“=:”。赋值号的左边是存储单元的名字，它一般是个变量的名字，当然也可以是数组元素名、记录域名和函数名，这些将在以后的有关章节中介绍。

赋值语句的作用是先计算“表达式”的值，然后将这个值存入由“名字”所指定的存储单元中。

在程序中，每个变量都是有类型的，表达式也有类型。在使用赋值语句时，一般要求赋值号左边的变量和赋值号右边的表达式有相同的类型。例如，整型表达式赋值给整型变量，实型表达式赋值给实型变量。例外的情况是，一个整型表达式也可以赋值给一个实型变量。

[例2.1] 设r是经过说明的实型变量，则下面两个赋值语句都是允许的，而且产生相同的效果。

```
r:=2.0*2.0;
r:=2*2
```

对于第一个赋值语句，先计算出实型表达式的值4.0，然后赋值给实型变量r。对于第二个赋值语句，先计算出整型表达式的值4，然后将它转化成实数4.0，再赋值给实型变量r。

注意：当程序段中有多个语句时，语句和语句之间要用分号“;”分隔。

赋值号右边的表达式中允许出现变量。

[例2.2]　下面这个赋值语句的作用是先计算出2倍的r值，然后将它存入与变量d相对应的存储单元中去。

```
d:=2*r
```

显然，为了保证在执行了这个赋值语句后，变量d有一个确定的值，首先应该保证在执行这个赋值语句前，变量r有确定的值。

赋值号右边的表达式中还可以出现赋值号左边的变量。

[例2.3]　下面这个赋值语句的作用是将存放在变量n中的值加1。

```
n:=n+1
```

在这个赋值语句中，出现在赋值号两边的n是同一个变量，对应同一个存储单元。

2.2　输入语句

在程序中给变量一个确定的值，除了用赋值语句外，还可以使用read和readln这两个输入语句。输入时，数据可以通过键盘输入，也可以从磁盘文件中读取。这里只介绍如何通过键盘输入数据（从磁盘文件中读取数据的方法将在第9章中介绍）。

2.2.1　read语句

read语句的格式为

```
read(变量1, 变量2, …, 变量n)
```

其功能是，从键盘读取n个数据，依次存入变量1，变量2，…，变量n中。

[例2.4]　设a,b,c都是整型变量。当执行到语句

```
read(a,b,c)
```

时，程序会等待你从键盘输入数据。如果输入

15　21　32

则a,b,c三个变量分别获得值15,21,32.

如果read语句要求从键盘读取n个数据，而实际输入的数据多于n个，则程序只读取前n个数据．剩余的数据可被后面的read语句或readln语句使用．

[例2.5]　设a,b,c都是整型变量．顺序执行语句

```
read(a);read(b,c)
```

若输入

15　21　32

则效果与例2.4相同．

如果read语句要求从键盘读取n个数据，而实际输入的数据少于n个，则程序会继续等待，直到读取了n个数据为止．

[例2.6]　当执行到语句

```
read(a,b,c)
```

时，用下面几种方法输入，其效果完全相同：

① 用一个输入行输入

15　21　32

② 分两个输入行输入

15　21

32

或者

15

21　32

③ 分三个输入行输入

15

21

32

从键盘输入的每个数据必须和read语句中对应的变量类型一致．

[例2.7]　设a是整型变量，b是实型变量．当执行到语句

```
read(a,b)
```

时，如果输入

15.5　31

则屏幕上会显示一条出错信息，因为15.5是个实数，不能存入整型变量a中．

从键盘输入整型或实型数据时，数据和数据之间要用空格作分隔符，

可以是一个空格，也可以是多个空格．但在输入字符型数据时，字符数据和字符数据之间不能有分隔符，而且字符数据不可以带引号．

[例2.8] 设a,b都是字符型变量．如果想通过执行语句

```
read(a,b)
```

把字符'A'存入变量a，把字符'B'存入变量b，则应在键盘上输入

```
AB
```

如果输入

```
A  B
```

则结果为：a='A'，b=' '；如果输入

```
'A' 'B'
```

则结果为：a=' '，b='A'．

2.2.2 readln语句

readln语句有两种格式．

格式1：

```
readln
```

格式2：

```
readln(变量1, 变量2, …, 变量n)
```

第一种格式readln语句的功能是执行一次换行操作，使下一个read或readln语句从下一个输入行中读取数据．

第二种格式readln语句的功能相当于顺序执行下面两个语句：

```
read(变量1, 变量2, …, 变量n);
readln
```

[例2.9] 执行下面3个语句

```
read(a,b);
readln;
read(c)
```

若分两个输入行输入

```
15    21    32
27
```

则语句read(a,b)只读取第一个输入行中的前面两个数据15和21，语句readln执行一次换行，使语句read(c)从第二个输入行中读取数据27．

在程序中只有一个输入语句的情况下，使用read语句或第二种格式的readln语句，将产生相同的结果。但是，当程序中有多个输入语句时，数据输入方式不同，有可能产生不同的结果。

[例2.10]　如果在键盘上输入

21　　32　　27

45　　15

则执行语句

read(a,b,c,d,e)

的结果和执行语句

readln(a,b,c,d,e)

的结果完全相同，都是

a=21　b=32　c=27　d=45　e=15

而执行语句

read(a,b);read(c,d)

和执行语句

readln(a,b);read(c,d)

的结果却不相同，前者结果是

a=21　b=32　c=27　d=45

后者的结果是

a=21　b=32　c=45　d=15

2.3　输出语句

要将程序处理结果输出，需要用输出语句。在Pascal语言中两个主要的输出语句是write语句和writeln 语句。输出时，结果可以显示在屏幕上，也可以保存到磁盘文件中。本章只介绍如何将结果显示在屏幕上（把结果保存到磁盘文件中去的方法将在第9章中介绍）。

2.3.1　write语句

write语句的格式为

write(输出项1, 输出项2, …, 输出项n)

其中，每个输出项可以是常量、变量或一般的表达式。其功能是，将各输

出项依次显示在屏幕上，当一行显示不下时，自动换行显示。在执行完一个write语句后，输出不会自动换行，下一个write语句的输出结果紧跟着输出，即输出在同一行上。

[例2.11] 执行语句

```
write('计算结果:',25*37)
```

将在屏幕上看到

```
计算结果:925
```

这个语句中有两个表达式。第一个是字符串，输出时照原样输出；第二个是算术表达式，输出时先计算出结果925，然后再输出。

[例2.12] 顺序执行语句

```
write('计算结果:');
write(25*37)
```

的结果和上例看到的完全相同。

2.3.2 writeln语句

writeln语句有两种格式。

格式1：

```
writeln
```

格式2：

```
writeln(输出项1, 输出项2, …, 输出项n)
```

第一种格式writeln语句的功能是在屏幕上执行一次换行操作，使下一个write或writeln语句从下一个输出行开始输出。

第二种格式writeln语句的功能相当于顺序执行下面两个语句：

```
write(输出项1,输出项2,…,输出项n);
writeln
```

[例2.13] 顺序执行以下3个语句：

```
write('计算结果:');
writeln;
write(25*37)
```

在屏幕上可以看到

```
计算结果:
925
```

2.3.3　输出格式

数据的输出格式有标准格式和自定义格式两种。

1. 标准格式

一个数据在输出时占多少位，与数据的类型有关。所谓标准格式，是指数据在输出时，所占的位数是由Pascal系统预先定义的。

integer类型的数据在输出时，按其实际位数输出；

real类型的数据在输出时，一律按指数形式输出，占17位；

char类型的数据在输出时，只占1位；

boolean类型的数据在输出时，占4位或5位。

[例2.14]　执行语句

```
writeln('计算结果:',1234)
```

将在屏幕上看到

```
计算结果:1234                    {占4位}
```

[例2.15]　执行语句

```
writeln('计算结果:',-123.4)
```

将在屏幕上看到

```
计算结果:-1.2340000000E+02       {占17位}
```

2. 自定义格式

自定义格式有

```
x:n
```

和

```
x:n1:n2
```

两种。其中第一种格式用于输出整型、字符型、布尔型和字符串数据；第二种格式用于输出实型数据。

在第一种格式中，正整数n规定了数据x输出时的位数。如果x的值不足n位，则输出时左边补空格，凑齐n位；如果x的值超过了n位，则按x的实际位数输出。

[例2.16]　执行语句

```
writeln('计算结果:',1234:5)
```

将在屏幕上看到

计算结果: 1234　　{占5位，左边补一个空格}

[例2.17]　执行语句

writeln('计算结果:',1234:2)

将在屏幕上看到

计算结果:1234　　{占4位，按实际值输出}

在第二种格式中，正整数n1规定了实数x输出时包括正负号和小数点在内的总位数；n2规定了小数部分的位数；包括正负号在内的整数部分的位数是n1–n2–1位。

如果x的整数部分的位数多于n1–n2–1位，则按实际位数输出整数部分；

如果x的整数部分的位数少于n1–n2–1位，则输出时左边补空格，凑齐n1–n2–1位整数部分；

如果x的小数部分的位数少于n2位，则输出时右边补0，凑齐n2位小数；

如果x的小数部分的位数多于n2位，则只输出小数点后n2位。

[例2.18]　执行语句

writeln('计算结果:',123.4:2:1)

将在屏幕上看到

计算结果:123.4　　{占5位，整数部分按实际位数输出}

[例2.19]　执行语句

writeln('计算结果:',123.4:7:1)

将在屏幕上看到

计算结果:　123.4　　{占7位，左边补两个空格}

[例2.20]　执行语句

writeln('计算结果:',123.4:8:2)

将在屏幕上看到

计算结果:　123.40　　{占8位，左边补两个空格，右边补一个0}

[例2.21]　执行语句

writeln('计算结果:',123.456:8:2)

将在屏幕上看到

计算结果:　123.45　　{占8位，左边补两个空格，只输出两位小数}

2.4　程序举例

下面，我们用所学的赋值语句、输入语句和输出语句来编写几个简单

的Pascal程序。这些简单的程序有个共同的特点：程序中各语句的执行顺序和它们的书写顺序完全相同。具有这种特点的程序称为顺序结构的程序。

[例2.22]　从键盘输入圆的半径r，计算圆直径d、圆周长c和圆面积a。

[程序]

```
program ch222;
 const pi=3.14;
 var r:real;
 begin
  write('input r:');              {提示用户输入圆半径}
  read(r);                        {输入圆半径}
  writeln('d=',2*r:6:2);          {输出圆直径}
  writeln('c=',2*pi*r:6:2);       {输出圆周长}
  writeln('a=',pi*r*r:6:2)        {输出圆面积}
 end.
```

[运行实例]

```
input r:5
d= 10.00
c= 31.40
a= 78.50
```

[例2.23]　从键盘输入a、b、c的值，用公式

$$F=\sqrt{s(s-a)(s-b)(s-c)}$$

计算出三角形的面积，其中a、b、c是三角形3条边的长度，s=(a+b+c)/2。

[程序]

```
program ch223;
 var a,b,c,s:real;
 begin
  write('input a,b,c:');
  read(a,b,c);
  s:=(a+b+c)/2;
  s:=sqrt(s*(s-a)*(s-b)*(s-c));
  write('s=',s:6:2)
 end.
```

[运行实例]

```
input a,b,c:3 4 5
```

s=　6.00

在这个程序中使用了求平方根的标准函数sqrt．此函数可以带一个值大于零的实型参数，函数值也是实型的．

[例2.24]　本例要说明,计算机不能精确表示实数,计算结果存在误差．

[程序]

```
program ch224;
var a,b,c,d:real;
begin
 a:=1.2345678909;
 b:=1.2345678901;
 writeln('a=',a);
 writeln('b=',b);
 writeln('a-b=',a-b);        {显然，a-b的值应该是8.0E-10}
 c:=0.12345678901234;
 d:=0.12345678901239;   {显然，c和d不相等}
 writeln;
 writeln('c=',c);
 writeln('d=',d)
end.
```

[运行结果]

```
a= 1.2345678909E+00
b= 1.2345678901E+00
a - b= 8.0035533756E-10
c= 1.2345678901E-01
d= 1.2345678901E-01
```

习　题　二

2.1　已知a,b是实型变量,c,d是整型变量,下列赋值语句哪些是对的?

(1) a:=c　　(6) a:=c/d

(2) c:=a　　(7) a:=c div d/b

(3) c:=b+d　　(8) a:=c/d div b

(4) a+b:=d　　(9) a:=c*d*b

(5) a:=a+1　　(10) c:=d+1.0

2.2　设x,y,z,m,n,p,r,s都已说明为整型变量,程序中有下列语句:

```
readln(x,y,z);
readln(m,n,p);
read(r,s);
```

若数据分4行输入:

```
234  746  936  835
72   84
801  402
105  216
```

则在执行了上述输入语句后,每个变量各取什么值?

2.3　设a,b,c已说明为实型变量,要使a=3.5,b=24.0,c=298.4,采用下面的语句读入a,b,c的值,应如何组织数据?写出各数据行.

(1) read(a);read(b);read(c)

(2) readln(a);readln(b);readln(c)

(3) readln;read(a,b,c)

(4) readln(a,b);readln(c)

(5) readln(a);read(b);readln(c)

(6) read(a);readln;read(b);readln;read(c)

2.4　设a,b,c,d,e,f,g均为整型变量,其值分别为2,4,6,8,10,12,14,请写出执行下面各组语句后的输出结果:

(1) write(a,b,c);write(d,e,f);write(g)

(2) writeln(a,b,c);writeln(d,e,f);writeln(g)

(3) write(a,b);writeln;write(c);writeln(d,e);write(f)

(4) writeln;write(a);write(b,c,d,e);writeln(f)

(5) write(a:5,b:4,c:3);writeln;write(d:2,e:1);writeln(f:5,g:3)

(6) writeln('a=',a,'b=',b);writeln('c=':5,c:1,'d=':5,d:1);
writeln('e=',e:2,'f=',f:4);writeln;write(g:1)

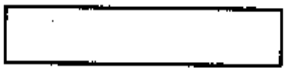
2.5　根据所给出的注释,将下面这个程序补充完整。

```
program change;
 var x,y:integer;  {用于存放两个整数}
 [          ]

 begin
  writeln('输入两个整数');
```

 {用read语句给变量x,y赋值}

write('输入的两个整数是:');

 {输出x,y}

 {交换变量x,y的值}

write('交换后,这两个整型变量的值是:');

 {输出x,y}

end.

2.6　写一个程序,它的功能是:先从键盘输入两个整型变量a,b的值,然后输出5个表达式

a+b

a–b

a*b

a div b

a mod b

的值．要求输出时，每个算式占一行．例如，若输入的两个整数是10和5，则要求输出:

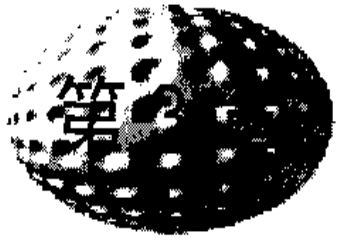
10+5=15

10–5=10

10*5=50

10 div 5=2

10 mod 5=0



程序的选择结构

在现实生活中，经常会遇到一些比较复杂的问题，要求我们“根据不同的情况，采取相应的措施”。在进行程序设计的时候，遇到类似的问题要使用选择结构。

[引例] 输入学生的百分制成绩，然后将它转换成等级制成绩：90至100为“优”；80至89为“良”；70至79为“中等”；60至69为“及格”；60以下为“不及格”。

[程序a]

```
program ch30a;
 var mark:integer;
 begin
  write('分数:');
  read(mark);
  if mark>=90
   then write('优')
  else                          {即mark<90}
   if mark>=80
   then write('良')
  else                          {即mark<80}
   if mark>=70
   then write('中等')
  else                          {即mark<70}
   if mark>=60
   then write('及格')
  else                          {即mark<60}
```

```
            write('不及格')
    end.
[程序b]
 program ch30b;
  var mark:integer;
  begin
   write('分数:');
   readln (mark);
   case mark div 10 of
    0..5:write('不及格');
       6:write('及格');
       7:write('中等');
       8:write('良');
    9,10:write('优')
   end;
  end.
```

这两个程序的功能是一样的．在第一个程序中，用if语句实现选择结构；在第二个程序中，用case语句实现选择结构。

Pascal语言用于实现选择结构的语句主要是if语句和case语句。在介绍这两种语句的使用方法之前，让我们先来学习一些布尔运算的知识。

3.1　布尔运算

判断一个条件“成立”或“不成立”的运算称为布尔运算。布尔运算的结果有“真”和“假”两种．“真”表示“条件成立”，“假”表示“条件不成立”．

布尔运算的概念和方法是由英国数学家布尔（Bale）最早提出的。实现布尔运算是计算机的一个非常重要的功能。

3.1.1　布尔值

在Pascal语言中，布尔运算结果的“真”和“假”用预定义标识符true和false表示：true表示“真”，false表示“假”．这两个值称为布尔值或布

尔常量.

3.1.2　布尔变量

布尔变量是用来存放布尔常量的。与其它类型的变量一样，布尔变量也必须先说明后使用。说明布尔变量要用预定义类型标识符boolean，例如：

```
var a:boolean;    {说明a是布尔变量}
    x:integer
```

在作了这个说明后，就可以将一个值为true或false的表达式赋值给变量a，例如：

```
a:=true;  {  a的值为true  }
a:=x>0  {  如果x大于0则a的值是true，否则a的值是false  }
```

注意：赋值语句a:=true 不可以写成a:='true'.

3.1.3　关系表达式

用关系运算符连结起来的式子称为关系表达式．关系表达式的值是个布尔值。

Pascal语言中的关系运算符共有7个：

<	>=	>	<=	=	<>	in
(小于)	(不小于)	(大于)	(不大于)	(等于)	(不等于)	(测试集合)

在第8章以前只使用前6个。

如何计算关系表达式的值呢?

对于整型或实型数据，根据运算符两边数据值的大小来确定关系表达式的值；对于字符型数据或枚举型数据（第5章中学习），则根据运算符两边数据值序号的大小来确定关系表达式的值.

例如，关系表达式 2>1 的值为true，因为2比1大；关系表达式 2<>1+1 的值为false，因为1加1等于2；关系表达式 'A'<'B' 的值为true，因为在字符集中，'A'的序号是65，而'B'的序号是66.

3.1.4　布尔表达式

用布尔运算符连结起来的式子称为布尔表达式．布尔表达式的值当然也是布尔值.

Pascal语言中的布尔运算符有3个：

not　　and　　or
(否定)　(并且)　(或者)

其中，not是“单目运算符”，只在运算符的后面有一个运算对象；and和or都是“双目运算符”，在运算符的前后各有一个运算对象。

[例3.1]　设x和y都是布尔类型的数据，则可以有布尔表达式：

not x　　{若x的值为true，则表达式的值为false；若x的值为false，则表达式的值为true}

x and y　{只有当x和y的值都为true时，表达式的值才为true；否则表达式的值为false　}

x or y　　{只有当x和y的值都为false时，表达式的值才为false；否则表达式的值为true}

在一个较为复杂的布尔表达式中可能包含多个布尔运算符。在这种情况下，应该先计算not，再计算and，最后计算or；对于相同的布尔运算符则从左往右计算；如果有括号则先计算括号里面的运算符。

[例3.2]　设x=true，y=false，z=false，则

x and y or not z　　值为true

not x or y and z　　值为false

not (not x or y)　　值为true

x and y and z or not x　　值为false

如果布尔运算符的运算对象是关系表达式，则关系表达式必须加括号，先进行关系运算，得到一个布尔值，然后再进行布尔运算。

[例3.3]　若要表示“a、b、c三个整数中任意两个之和都大于第三个”这样的条件，可用下面的布尔表达式：

```
(a+b>c)and(a+c>b)and(b+c>a)
```

或

```
not((a+b<=c)or(a+c<=b)or(b+c<=a))
```

3.1.5　布尔型数据的输入和输出

布尔型数据可以用write或writeln语句输出，但不可以用read或readln语句输入。

[例3.4]　设x=true，y=false，则程序段

```
writeln(x,' ',y);
writeln(true,' ',false)
```

.的执行结果为

```
true false
true false
```

[例3.5]　若x是布尔变量，则语句

```
read(x)
```

是错误的。

如果想在程序执行过程中，通过read或readln语句给布尔变量x赋值，则可利用条件语句进行间接输入。

[例3.6]　利用条件语句，间接输入布尔值。

```
var x:boolean;
    c:char;
    …
    readln(c);
    if (c='t')or(c='T')
    then x:=true
    else x:=false;
    …
```

当程序执行到语句readln(c)时，系统等待从键盘输入。此时，无论是输入一个字符还是输入多个字符，只要第一个字符是't'或'T'，变量x都将得到布尔值true；否则，变量x得到布尔值false。

3.2　if语句

if 语句有以下两种格式。

格式1：

```
if 条件 then 语句
```

格式2：

```
if 条件 then 语句 else 语句
```

其中，条件是个布尔变量、关系表达式或一般的布尔表达式；if、then、else都是保留字；then后面的语句称为then子句；else后面的语句称为else子句。

第一种格式的if语句的功能：如果条件成立，则执行then子句，否则就什么都不做。

第二种格式的if语句的功能：如果条件成立，则执行then子句，否则就执行else子句。

注意：

① 在第二种格式的if语句中,then子句的后面不能有分号。

② then子句和else子句都只能是一个语句。如果问题比较复杂，在某种条件下需要执行一组语句，则可以用begin和end将这组语句括起来，构成复合语句。按Pascal语法，不管复合语句中包含了多少个语句，复合语句本身只算一个语句。

[例3.7]　输入x，根据下面的公式计算并输出y值。

$$y=\begin{cases}0.9x & x\leqslant 100\\ 0.9x+20 & x>100\end{cases}$$

[程序]

```
program ch37;
 var x,y:real;
 begin
  write('x= ');
  readln(x);
  if x<=100
  then y:=0.9*x  {注意：这里不能有分号，因为这是第二种格式的if}
  else y:=0.9*x+20; {语句中的then子句}
  writeln('y= ',y)
 end.
```

[例3.8]　输入两个整数，将较大的放在变量max 中，较小的放在变量min中，然后输出max,min的值.

[解题思路]

输入两个整数到变量max和min中，如果max<min就交换max和min的值。为了完成交换，要用到第三个变量：先将第一个变量的值存入第三个变量，然后将第二个变量的值存入第一个变量，最后将第三个变量中的值存入第二个变量.

[程序]

```
program ch38;
 var max,min,t:integer;
 begin
  write('INPUT: ');
```

```
  readln(max, min);
  if max<min then
  begin
   t:=max;
   max:=min;
   min:=t
  end;
  writeln('OUTPUT:',max,min)
 end.
```

在这个程序中，then子句是个复合语句，包含了3个赋值语句，其作用是交换max和min这两个变量的值．

then子句和else子句可以是任何语句，包括if语句．若在if语句中还包含if语句，则称之为if语句的嵌套．

[例3.9]　输入x，根据下面的公式计算并输出y值．

$$y=\begin{cases}0.9x & x\leqslant 100\\ 0.8x+10 & 100<x\leqslant 300\\ 0.7x+5 & x>300\end{cases}$$

[程序a]

```
 program ch39a;
  var x,y:real;
  begin
   write('x= ');readln(x);
   if x<=100
   then y:=0.9*x
   else                {此时x一定大于100}
    if x<=300
    then y:=0.8*x+10
    else y:=0.7*x+5;
   writeln('y= ',y)
  end.
```

在这个程序中有两个if语句，其中一个if语句是另一个if语句的else子句．

对于例3.9，还可以写出其它形式的程序．例如：

[程序b]

```
 program ch39b;
```

```
var x,y;real;
begin
 write('x= ');readln(x);
 if x>100
 then
  if x>300
  then y:=0.7*x+5
  else y:=0.8*x+10
 else y:=0.9*x;
 writeln('y= ',y)
end.
```

在这个程序中也有两个if语句，其中一个if语句是另一个if语句中的then子句。

[程序c]

```
program ch39c;
 var x,y;real;
 begin
  write('x= ');readln(x);
  y:=0.9*x;
  if x>100
  then
   if x<=300
   then y:=0.8*x+10
   else y:=0.7*x+5;
  writeln('y= ',y)
 end.
```

这个程序在形式上和前面两个有较大区别，但功能相同．它先是给y赋值0.9x，然后判断条件x>100是否成立，若不成立，则不再对y重新赋值；若成立，则根据条件x<=300是否成立决定给y重新赋值0.8x+10或0.7x+5．

[程序d]

```
program ch39d;
{这是一个语法正确但不合题意的程序}
 var x,y;real;
```

```
begin
 write('x= ');readln(x);
 y:=0.9*x;
 if x<=300
 then
  if x>100
  then y:=0.8*x+10
 else y:=0.7*x+5;
 writeln('y= ',y)
end.
```

程序d和程序c形式上相似，程序设计者的意图是：先给y赋值0.9*x，然后判断条件x<=300是否成立，若不成立，说明x>300，就给y重新赋值0.7*x+5；若成立，则进一步判别x的值，此时，若x>100，则给y重新赋值0.8*x+10。可是，计算机并不是这样理解的。计算机认为程序d和程序c在结构上是完全相同的：

```
y:=0.9*x;
if x<=300
then
 if x>100
 then y:=0.8*x+10
 else y:=0.7*x+5
```

即，先给y赋值0.9x，然后判断条件x<=300是否成立，若不成立，则不对y重新赋值；否则，根据条件x>100是否成立决定对y重新赋值0.8x+10或0.7x+5。显然，这种计算方法不是题目所要求的。

注意：Pascal语法规定，在嵌套的if语句中，else子句总是与前面离它最近的then子句配对。

[例3.10] 邮局对邮寄包裹有如下规定：若包裹的长、宽、高中的任一尺寸超过1m，或重量超过30kg，均不予邮寄；对可以邮寄的包裹根据下表按重量计算邮资，并且每件收手续费0.2元。

重量(kg)	300km内收费标准(元/kg)	300km外收费标准(元/kg)
z<10	0.80	1.00
10<z<20	0.75	0.90
20<z<30	0.70	0.80

输入包裹尺寸、重量和邮寄距离，计算邮费，对不能邮寄的包裹给出相应信息。

[程序]

```
program ch310;
  const a=0.2; { 手续费 }
    var c,k,g,z,j,y:real; { 长,宽,高,重,距离,邮资 }
  begin
    write('input c,k,g,z,j:');
    readln(c,k,g,z,j);
    if (c>1)or(k>1)or(g>1)or(z>30)
    then writeln('can not send by post')
    else { 计算邮资 }
    begin
     if j<=300 { 300公里内 }
     then
      if z<=10
      then y:=0.80
      else if z<=20
             then y:=0.75
             else y:=0.70
     else{300公里外}
      if z<=10
      then y:=1.00
      else if z<=20
             then y:=0.90
             else y:=0.80;
     y:=y*z+a;
    writeln('cost=',y:7:2)
    end
  end.
```

3.3　case语句

case语句的格式为

```
case表达式of
 常量表1:语句1;
 常量表2:语句2;
 …
 常量表n:语句n
end
```

其中，case、of和end是保留字；表达式可以是整型、字符型、布尔型或枚举型，但不能是实型；每个常量表中可以只有一个常量，也可以有多个常量，常量和常量之间用逗号分隔，每个常量都是表达式的可能取值；跟在每个常量表后面的语句，可以是一个简单的语句，也可以是一个复合语句。

case语句的功能：计算表达式的值，如果这个值出现在某个常量表中，则执行跟在这个常量表后面的语句，其它语句不执行；如果这个值在任何一个常量表都没有出现，则什么都不做。

常量表的排列次序可以是任意的，不一定要从小到大排列，但各常量彼此应不相同。

[例3.11]　输入两个不为零的实数和一个算术运算符，输出这个算术表达式的值。

[程序]

```
program ch311;
 var x,y,s:real;
     c:char;
 begin
  write('INPUT x,y,c: ');
  readln(x,y);readln(c);
  case c of
   '+':s:=x+y;
   '-':s:=x-y;
```

```
    '*':s:=x*y;
    '/':s:=x/y
  end;
  writeln(x,c,y,'=',s)
end.
```

如果若干值连续的常量出现在同一个常量表中，则可以简化这个常量表：只写出最小的和最大的两个常量，中间用两个小圆点分隔。例如，常量表

```
0,1,2,3,4,5
```

可简写成

```
0..5
```

[例3.12]　计算某年某月有多少天。

[分析]

1月、3月、5月、7月、8月、10月和12月每月都有31天；4月、6月、9月和11月每月都有30天；2月平年有29天，闰年有28天。

如果年号能被400整除，则是闰年；如果年号能被4整除而不能被100整除，则也是闰年；其余情况都是平年。

[程序]

```
program ch312;
 var year,month,days:integer;
 begin
  write('INPUT year,month:');
  readln(year,month);
  if year<0
  then writeln('ERROR!')
  else begin
   case month of
    1,3,5,7,8,10,12:days:=31;
    4,6,9,11:days:=30;
    2:if (year mod 400=0) or
        (year mod 4=0) and (year mod 100<>0)
      then days:=29
      else days:=28
  end;  {case}
```

```
    writeln('days=',days)
   end    {else}
  end.
```

[例3.13]　编写计算下列分段函数的程序.

$$y=\begin{cases}\sin(ax) & 0.5\leqslant x<1.5\\ \cos(bx) & 1.5\leqslant x<2.5\\ \log_{10}(a/x) & 2.5\leqslant x<3.5\\ (bx)^{2/3} & 3.5\leqslant x<4.5\end{cases}$$

[程序]

```
program ch313;
 var a,b,x:real;
 begin
  write('INPUT a,b,x');
  readln(a,b,x);
  case round(x) of
   1: y:=sin(a*x);
   2: y:=cos(b*x);
   3: y:=ln(a/x)/ln(10);
   4: y:=exp(2/3*ln(b*x))
  end;
  writeln('y=',y:8:2)
 end.
```

在这个程序中使用了5个标准函数（round,sin,cos,exp和ln）。

round(x)是舍入函数，其中，x可以是实型表达式，函数值是最接近x的整数。例如，

round (0.7) =1

round (–0.7) = –1

sin(x)和cos(x)分别用来计算x的正弦值和余弦值，其中，x的单位是弧度。

ln(x)用于计算x的自然对数值，其中，x是实型表达式，自然对数的底是e。

如果要计算以10为底的对数$\log_{10}x$，可用换底公式

$$\log_{10} x = \frac{\ln(x)}{\ln(10)}$$

exp(x)用于计算e^x值，其中，e是自然对数ln的底。

Pascal语言中没有直接计算x^n的标准函数，但可以用下面公式来求x^n的值：

$$x^n = e^{n \times \ln(x)}$$

[例3.14] 本例要说明:两个理论上不相等的实数可能会被计算机认为相等;两个理论上相等的实数也可能会被计算机认为不相等。

[程序]

```
program ch314;
 var a,b,c,d:real;
 begin
  a:=0.12345678901234;
  b:=0.12345678901239;{显然,a不等于b}
  c:=0.12345678909;
  d:=0.12345678901;
  e:=8.0E-11;
  f:=c-d;                    {显然,e等于f}
  if a=b
  then writeln('a等于b')
  else writeln('a不等于b')
  if e=f
  then writeln('e等于f')
  else writeln('e不等于f')
 end.
```

[运行结果]

a等于b

e不等于f

注意：在程序中应尽量避免对两个实数进行相等或不相等的比较。

3.4 两种选择结构的对比

if语句和case语句都可以用来实现选择结构。一般地说，if语句用于实

现二路选择，即根据布尔类型的条件的值，来决定要不要执行某个语句(或执行两个语句中的哪一个)，如图3.1所示;case语句用于实现多路选择，即根据有序类型的表达式的值，从多个语句中选择一个语句执行(或一个都不执行)，如图3.2所示。

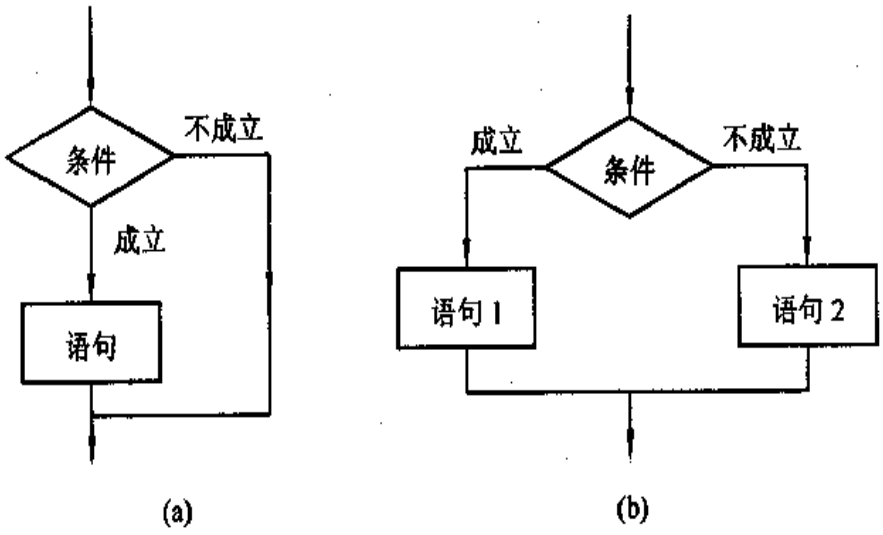


图3.1　二路选择

(a) if_then_结构　(b) if_then_else结构

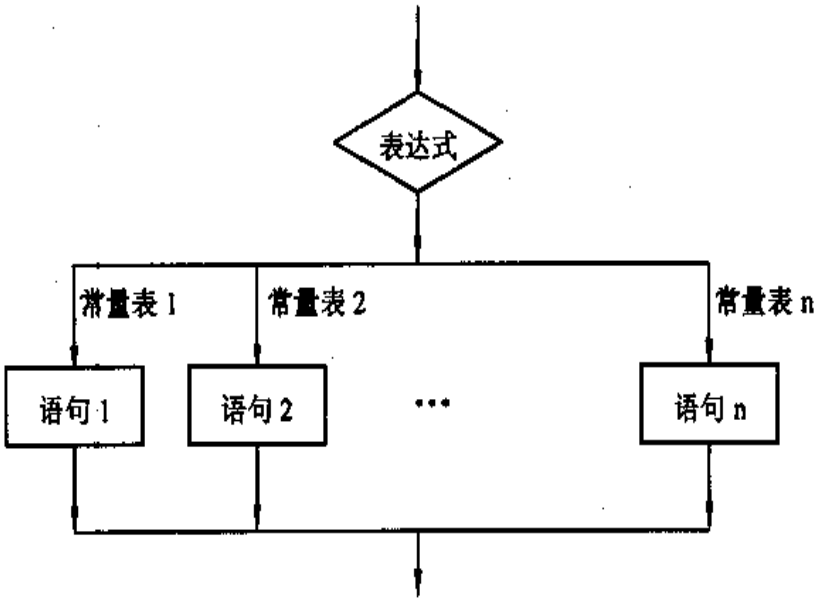


图3.2　多路选择

用嵌套的if语句也可以实现多路选择，例如：

```
if 表达式=常量1
 then 语句1
else if 表达式=常量2
```

```
 then 语句2
else if 表达式=常量3
 then 语句3
else if 表达式=常量4
 then 语句4
else if 表达式=常量5
 then 语句5
```

但是，其可读性显然不如case语句(假定表达式是有序类型的)：

```
case 表达式 of
 常量1: 语句1;
 常量2: 语句2;
 常量3: 语句3;
 常量4: 语句4;
 常量5: 语句5
end
```

为了提高程序的可读性,应该尽量用case语句来实现多路选择。

习　题　三

3.1　填空题。

(1) 判断一个条件是否成立的运算称为_____运算。

(2) 用_____连结起来的式子称为关系表达式。

(3) 用_____连结起来的式子称为布尔表达式。

(4) 用begin和end括起来的一组语句称为_____语句。

(5) 所谓嵌套的if语句是指_____。

(6) 在嵌套的if语句中，else子句总是与_____配对。

(7) 实现多路选择应尽量使用_____语句。

(8) 实现二路选择一般使用_____语句。

(9) 布尔运算符有__个，其中____是单目运算符。

(10) 和if语句if c='c' then b:=true else b:=false 等价的赋值语句是_____。

3.2　选择题。

(1) 要实现选择结构：

“如果a不小于b并且c等于d，则把w的值放入u中；否则,把y的值放入x

中。”可以使用语句_____。

A.
```
if a<b then
  if c=d
  then x:=y
  else u:=w
```

B.
```
if a<b then
  x:=y
else if c=d then
  u:=w
else x:=y
```

C.
```
if a<b then
  begin
   if c=d then
    x:=y
   else u:=w
  end
```

D.
```
if a<b then
  begin
  if c=d then
   x:=y
  end
  else u:=w
```

(2) 设n和x均为整型变量，则与嵌套的if语句

```
if(n<10)and(n>0)then
 if n>5
 then if n<8
  then x:=0
  else x:=1
 else if n>2
  then x:=3
  else x:=4
```

等价的case语句是_____。

A.
```
case n of
  1,2   :x:=4;
  3,4,5 :x:=3;
  8,9   :x:=1;
  6,7   :x:=0
  end
```

B.
```
case n of
  1       :x:=4;
  2,3,4,5 :x:=3;
  8,9     :x:=1;
  6,7     :x:=0
  end
```

C.
```
case n of
  1,2   :x:=4;
  3,4,5 :x:=3;
  9     :x:=1;
  6,7,8 :x:=0
```

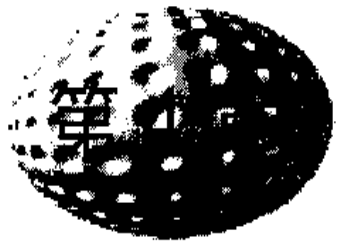
D.
```
case n of
  1,2,3 :x:=4;
  4,5   :x:=3;
  8,9   :x:=1;
  6,7   :x:=0
```

```
end                    end
```

3.3　编程题。

(1) 输入3个整数，将它们按从小到大的顺序输出。

(2) 用case语句实现例3.10所要求的功能。



程序的循环结构

循环结构用于描述那些需要重复处理的问题。

[引例] 输入全班30个同学某课程的考试分数，求全班平均分。

用s表示全班总分，设其初值为0；用x表示某个学生的分数。把语句组

```
read(x); s:=s+x
```

重复执行30次，求得全班总分，最后输出平均分s/30。

显然，用顺序结构把语句组 read(x); s:=s+x 连续写上30遍是很乏味的。如果采用循环结构则非常简单：

[程序]

```
program ch40;
 const n=30;
 var s,x,i:integer;
 begin
  s:=0;
  for i:=1 to n do
  begin
   read(x);s:=s+x
  end;
 write(s/30:6:2)
end.
```

在这个程序中，第6行到第9行就是一个循环结构，重复执行的对象（称为循环体）是第7行到第9行的一个复合语句。重复执行次数由第6行的for语句控制，变量i称为循环控制变量，从1变化到30，表示循环体要重复执行30次。

Pascal语言用于实现循环结构的语句除了for语句外，还有while语句和

repeat语句。本章详细介绍这3种语句的用法。

4.1 while语句

while语句的格式为

```
while 循环继续条件 do 循环体
```

其中，while和do是保留字；循环继续条件可以是一个布尔常量、布尔变量、关系表达式或布尔表达式；循环体只能是一个语句（包括复合语句）。

while语句的执行步骤是

① 判断循环继续条件是否成立，若成立则转至②，否则转至③；

② 执行一遍循环体，然后返回①；

③ 执行循环体后面的其它语句（如果有的话）。

图4.1描述了while语句的执行过程。

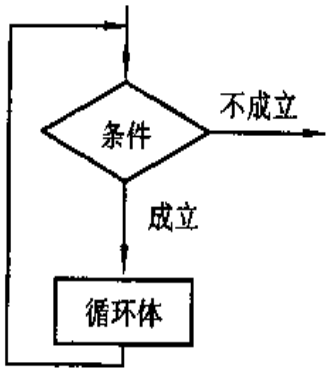


图4.1 while语句执行过程

在使用while语句时，要注意：

① 为了能使循环正常终止，循环体中一定要有能使循环继续条件变为不成立的语句，否则循环永不结束，出现**死循环**。

② 由于是“先判断后执行”，所以，循环继续条件中所含变量必须在while语句之前先赋初值。

[例4.1] 从键盘读入若干字符，分别统计其中字母字符和数字字符的个数，当读到字符'#'时，结束并输出统计结果。

[解题思路]

用m和n分别表示字母字符和数字字符的个数，它们的初值都为0。

读一个字符到变量ch中，在ch<>'#'的情况下，重复执行下列操作：

- 若ch是字母，则m加1；

- 若ch是数字，则n加1；
- 读一个字符到ch中。

当读到字符'#'时，循环结束，输出m,n值。

[程序]

```
program ch41;
 var m,n:integer;
     ch:char;
 begin
  m:=0; n:=0;
  read(ch);
  while ch<>'#' do
  begin
   case ch of
    'a'..'z',
    'A'..'Z':m:=m+1;
    '0'..'9':n:=n+1
   end;
   read(ch)
  end;
  writeln('m=',m);
  writeln('n=',n)
 end.
```

[运行实例]

```
ui78g6yt67gfy#
m=8
n=5
```

[例4.2]　输入一系列正整数，找出其中最大的.

[解题思路]

用变量max 表示最大的正整数，用输入一个负数作为循环结束的条件。输入第一个数据，将它赋值给max；在输入的数据是正数的情况下，重复执行下列操作：输入下一个数据，如果它大于max，就更新max值。

当循环结束时，如果max>=0，则说明至少输入了一个正数，最大值已存入max,可输出max；如果max<0，则说明一个正数也没有输入过，什么都不输出。

[程序]

```
program ch42;
 var x,max:integer;
 begin
  read(x);
  max:=x;
  while x>=0 do
  begin
   read(x);
   if x>max
   then max:=x
  end;
  if max>=0 then
   writeln('max=',max);
end.
```

[运行实例]

```
67  75  43  789  432  65  -9
max=789
```

[例4.3]　输入一个正整数，求它的各位数字之和。

[解题思路]

把这个正整数存入变量x中；用s表示各位数字之和，其初值为0。

在x<>0的情况下，重复执行下列操作：用10除x，把余数累加到变量s中，把商放回到变量x中。

循环结束后输出s值。

[程序]

```
program ch43;
 var s,x:integer;
 begin
  write('input x:');
  read(x);
  s:=0;
  while x<>0 do
  begin
   s:=s+x mod 10; {加上一个余数}
```

```
    x:=x div 10   {更新x}
  end;
  writeln('s=',s)
 end.
```

[运行实例]

```
 input x:586
 s=19
```

[例4.4]　利用格里高利公式

$$\frac{\pi}{4}=1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\cdots$$

求π的近似值，直到最后一项的绝对值小于1E-4为止.

[解题思路]

先对等号右边的式子求和，然后再乘以4即为π的值．等号右边的求和式子有这样的规律：每一项的分母是前一项分母加2；每一项的符号是正负交替。

用p表示求和结果，s表示分子，n表示分母．一开始，令n=1, s=1, p=1; 然后，在s/n的绝对值不小于1E-4的情况下，重复执行下列操作：

- 分母n加2；
- 分子改变正负号；
- 把s/n累加到变量p中．

循环结束后，将p的值乘以4．

求s/n的绝对值可用标准函数abs(s/n)．

[程序]

```
program ch44;
 var n,s:integer;
     p :real;
 begin
  n:=1; s:=1;
  p:=1;
  while abs(s/n)>=1E-4 do
  begin
   n:=n+2;     {分母加2}
   s:= -s;     {分子变符号}
   p:=p+s/n    {累加}
```

```
    end;
    p:=4*p;
    writeln;
    write('p=',p)
  end.
```

[运行结果]

P= 3.1417926136E+00

4.2　repeat语句

repeat语句的格式为

```
repeat
  循环体
until 循环结束条件
```

其中，repeat和until是保留字；循环结束条件可以是布尔常量、布尔变量、关系表达式或布尔表达式；循环体可以是一个语句，也可以是用分号分隔的多个语句。

repeat语句的执行步骤是

① 执行一遍循环体；

② 判断循环结束条件是否成立，若成立，则转至③，否则转至①；

③ 执行后面的其它语句（如果有的话）。

图4.2描述了repeat语句的执行过程。

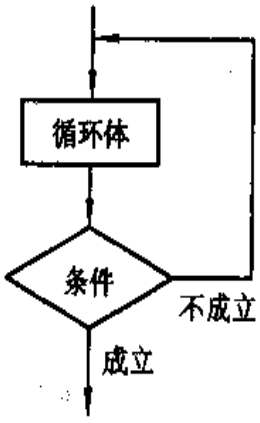


图4.2　repeat语句执行过程

在使用repeat语句时，要注意：

(1) 为了避免出现死循环，循环体中一定要有能使循环结束条件变为成立的语句。

(2) 由于是“先执行后判断”，所以，无论循环结束条件是否成立，循环体至少要执行一遍。

[例4.5]　输入一个实数m，计算并输出满足条件

$$1+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{n}>m$$

的n值。

[解题思路]

用s表示前n项之和，第n项的值为1/n，s,n的初始值均为0。

重复执行下列操作，直到s>m为止：n加上1；s加上1/n.

[程序]

```
program ch45;
 var n:integer;
     s,m:real;
 begin
  s:=0; n:=0;
  write('input m:');
  read(m);
  repeat
   n:=n+1;
   s:=s+1/n
  until s>m;
  writeln('n=',n)
 end.
```

[运行实例]

```
input m:5
n=83
input m:10
n=12367
```

[例4.6]　在Pascal语言中，整型变量的最大值是32767，如果将一个大于32767的数存入一个整型变量，那么，这个变量中实际保存的将是一个小

于32767的值。

根据这个特性，写一个程序，对于满足条件n!＜32767的一切n，计算并输出n!的值。

n!读作“n的阶乘”，定义如下：

n!=1×2×3×4×…×n

[解题思路]

用p1表示i!，用p0表示(i-1)!；p1和i的初值均为1。

重复执行下列操作，直到p1<p0为止：

- 输出p1；
- i加1；
- 把p1的值存入p0中；
- p1乘上i。

[程序]

```
program ch46;
 var p0,p1,i:integer;
 begin
  p1:=1; i:=1;
  repeat
   writeln(i,'!=',p1); {输出i的阶乘}
   i:=i+1;             {准备计算i+1的阶乘}
   p0:=p1;             {保存i的阶乘}
   p1:=p1*i            {计算i+1的阶乘}
  until p1<p0
 end.
```

[运行结果]

```
1!=1
2!=2
3!=6
4!=24
5!=120
6!=720
7!=5040
```

[例4.7]　利用台劳公式

$$e=1+\frac{1}{1!}+\frac{1}{2!}+\frac{1}{3!}+\cdots$$

求e的近似值，直到最后一项小于1E-7为止。

[解题思路]

显然，公式中第i-1项的值若等于p，则第i项的值就是p/i。

用变量e表示公式前i项之和，p表示第i项的值，e,i,p的初始值均为1。

重复执行下列操作，直到p<1E-7为止：

- 将p值改为p/i；
- 将p累加到e中；
- i加1。

[程序]

```
program ch47;
  var e,p:real;
      i :integer;
  begin
   e:=1; p:=1;
   i:=1;
   repeat
    p:=p/i;  {求得下一项}
    e:=e+p;  {累加}
    i:=i+1   {准备求下一项}
   until p<1E-7;
   writeln;
   write('e=',e)
  end.
```

[运行结果]

```
e= 2.7182818262E+00
```

4.3 for语句

for语句有两种格式。

格式1：

```
for 循环控制变量 := 初值 to 终值 do 循环体
```

格式2:

```
for 循环控制变量 := 初值 downto 终值 do 循环体
```

其中，for、to、downto和do 都是保留字；循环控制变量可以是整数类型、字符类型等有序类型的简单变量；初值和终值可以是和循环控制变量类型相同的表达式；循环体只能是一个语句（包括复合语句）。

第一种格式的for语句称为递增型for语句，其执行步骤如下：

① 将初值赋值给循环控制变量；

② 判断条件“循环控制变量的值大于终值”是否成立，若成立则转至⑥；

③ 执行一遍循环体；

④ 将循环控制变量的值改为 succ(循环控制变量)；

⑤ 转至②；

⑥ 执行循环体后面的其它语句（如果有的话）。

在步骤④中，succ(循环控制变量) 表示取循环控制变量的后继值．这是一个标准函数，其参数必须是有序类型的数据，函数值是参数的“下一个值”。例如，

```
succ(6)=7;
succ(-4)=-3;
succ('y')='z'
```

图4.3(a)描述了递增型for语句的执行过程。

第二种格式的for语句称为递减型for语句，其执行步骤如下：

① 将初值赋值给循环控制变量；

② 判断条件“循环控制变量的值小于终值”是否成立，若成立则转至⑥；

③ 执行一遍循环体；

④ 将循环控制变量的值改为 pred(循环控制变量)；

⑤ 转至②；

⑥ 执行循环体后面的其它语句（如果有的话）。

在步骤④中，pred(循环控制变量) 表示取循环控制变量的前趋值．这是一个标准函数，其参数必须是有序类型的数据，函数值是参数的“前一个值”。例如，

```
pred(6)=5;
pred(-4)=-5;
```

pred('y')='x'

图4.3(b)描述了递减型for语句的执行过程。

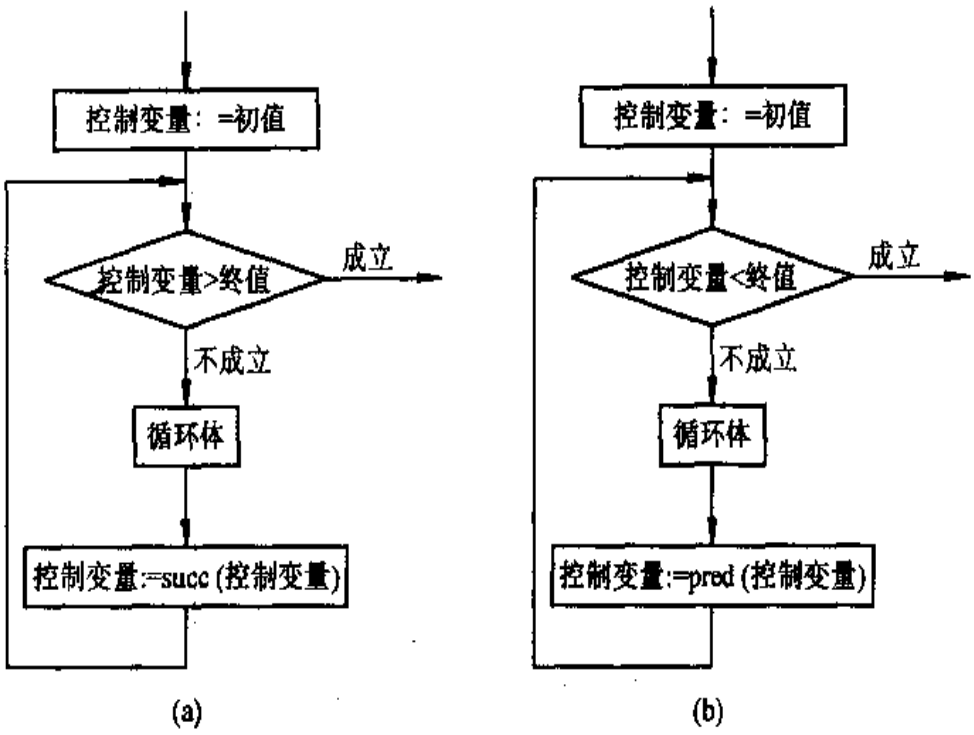


图4.3　for语句执行过程

(a) 递增型　(b) 递减型

[例4.8]　写一个程序，按正、反两种顺序输出26个大写英文字母。

[解题思路]

由于'A'<'Z',因此,正序输出要用递增型for语句，反序输出要用递减型for语句。

[程序]

```
program ch48;
 var c:char;
 begin
  for c:='A' to 'Z' do write(c);
  writeln;
  for c:='Z' downto 'A' do write(c)
 end.
```

[运行结果]

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ZYXWVUTSRQPONMLKJIHGFEDCBA
```

[例4.9] 找出所有满足关系

(AB+CD)(AB+CD)=ABCD

的四位整数ABCD.

[解题思路]

设m和n分别是4位整数i的高两位和低两位，则

m=i div 100

n=i-m*100

对每一个4位整数i，分离出它的高两位m和低两位n，若条件(m+n)(m+n)=i成立，则输出i值.

[程序]

```
program ch49;
 var i,m,n:integer;
 begin
  for i:=1000 to 9999 do
  begin
   m:=i div 100;    {分离出高两位}
   n:=i-m*100;      {分离出低两位}
   if i=sqr(m+n) then writeln(i)
  end
 end.
```

[运行结果]

```
2025
3025
9801
```

在这个程序中出现的标识符sqr 是一个标准函数名. 这个标准函数的参数可以是整型或实型的数据，函数值等于参数值的平方. 例如,

sqr(3)=9

sqr(3.0)=9.0

在使用for语句时，要注意：

① 初值和终值在开始重复之前计算一次，在重复执行过程中始终保持不变；

② 在循环体中不能有对循环控制变量赋值的语句。

[例4.10] 这个例子说明,在循环体中修改终值不影响循环的执行次数.

[程序]

```
program ch410;
{想得到输出结果:12345678}
var i,n:integer;
begin
 n:=5;
 for i:=1 to n do
 begin
  write(i);
  if i=5 then n:=8  {修改终值}
 end
end.
```

[运行结果]

12345

[例4.11] 这个例子说明,在循环体中不可以对循环控制变量重新赋值.

[程序]

```
program ch411;
{想得到输出结果:13579}
var i,n:integer;
begin
 n:=9;
 for i:=1 to n do
 begin
  write(i);
  i:=i+1  {给循环控制变量多加一个1}
 end
end.
```

[运行结果]

出现死循环.

4.4 多重循环

在某个循环语句的循环体中还可以出现循环语句，这种循环套循环的

程序结构称为多重循环。

[例4.12]　输出数字三角形

```
    1
   121
  12321
 1234321
123454321
```

[解题思路]

用m表示每一行中第一个数字前面的空格数，用i控制输出的行数，每输出一行m减1。

输出第i行可由以下4步实现：

(1) 输出m个空格；

(2) 输出数字1至i；

(3) 输出数字i-1至1；

(4) 换行。

[程序]

```
program ch412;
 const
  n=5;  {行数}
  p=10; {中心点到屏幕左边界的距离}
 var i,j,k,m:integer;
 begin
  m:=p;                {准备输出第一行}
  for i:=1 to n do
  begin
   write(' ':m);  {输出m个空格}
   for j:=1 to i do
    write(j);      {输出数字1至i}
   for j:=i-1 downto 1 do
    write(j);      {输出数字i-1至1}
   writeln;          {换行}
   m:=m-1          {准备输出下一行}
  end
 end.
```

[例4.13] 输出一张乘法口诀表。

[解题思路]

用i控制行数，用j控制每一行上的算式数，i从1到9，j从1到i，每输出一个算式 j*i后空两格。

[程序]

```
program ch413;
 var i,j:integer;
 begin
  writeln;
  for i:=1 to 9 do  {输出9行}
  begin
   for j:=1 to i do  {输出第i行}
    write(j,'*',i,'=',i*j:2,'  ');
   writeln
  end
 end.
```

[运行结果]

```
1*1=1
1*2=2 2*2= 4
1*3=3 2*3= 6  3*3= 9
1*4=4 2*4= 8  3*4=12 4*4=16
1*5=5 2*5=10 3*5=15 4*5=20 5*5=25
1*6=6 2*6=12 3*6=18 4*6=24 5*6=30 6*6=36
1*7=7 2*7=14 3*7=21 4*7=28 5*7=35 6*7=42 7*7=49
1*8=8 2*8=16 3*8=24 4*8=32 5*8=40 6*8=48 7*8=56 8*8=64
1*9=9 2*9=18 3*9=27 4*9=36 5*9=45 6*9=54 7*9=63 8*9=72 9*9=81
```

4.5 3种循环语句的对比

while语句、repeat语句和for语句都是用来实现循环结构的．它们功能相近，但不完全等效。

while语句和repeat语句用于实现条件型循环：用条件来控制是否要执行一遍循环体。

for语句用于实现计数型循环：通过计算循环控制变量初值和终值的差来决定执行多少遍循环体。

一般地说，如果事先知道循环的确切遍数，就尽量使用for语句；否则就只能使用while语句或repeat语句。

从功能上说，while语句和repeat语句的区别在于：由while语句控制的循环体可能一遍都不执行；而由repeat语句控制的循环体则至少要执行一遍。

习 题 四

4.1 填空题。

(1) 实现循环结构的语句主要有_____语句、_____语句和_____语句3种。

(2) 如果语句while T1 do S1 和语句repeat S2 until T2 功能相同,则T1和T2的关系是_____。

(3) 为了避免出现死循环，在while语句的循环体中要有_____的语句，在repeat语句的循环体中要有_____的语句。

(4) 循环体至少执行一遍的循环语句是_____语句。

(5) for语句有_____型和_____型两种。

(6) 循环体中又出现循环语句，这种程序结构称为_____。

(7) 实现计数型循环一般用_____语句。

(8) 实现条件型循环一般用_____语句和_____语句。

(9) 在for语句中，循环控制变量的初值和终值是在_____计算的。

(10) 在for语句的循环体中，不可以有对_____赋值的语句。

4.2 阅读下列不完整的程序,根据程序功能，将它们补充完整。

(1)

```
program p421;
{输入一系列正整数,找出其中最小的}
 var x,min:integer;
 begin
 [          ]
  repeat
```

```
        [          ]
          if x<min then [          ]
        until [      ] ;  {用输入一个负数来控制循环结束}
         write(min)
        end.
```

(2)

```
  program p422;
  {求100以内的正的奇数之和}
   var x,s:integer;
   begin
    [          ]
     while x<100 do
    [          ]
     write(s)
    end.
```

(3)

```
  program p423;
  {顺序输出序列0,1,1,2,3,5,8,13,21,... 中值小于100的那些项}
   var a,b,c :integer;  {a表示b的前一项,c表示b的后一项}
   begin
    [          ]
     write(a,'  ');
     while b<100 do
     begin
      write(b,'  ');
      c:=a+b;
      [          ]
     end
    end.
```

(4)

```
program p424;
{找到并输出被2和3除余数都是1的前20个正整数}
 var i,x:integer;
 begin
 [          ]
  repeat
   x:=x+1;
   if (x mod 2=1) and [          ]
   then [          ]
  until i=20
 end.
```

(5)

```
program p425;
{求序列1,1/2,1/3,1/4,…的前10项之和}
 var i:integer;s:real;
 begin
 [          ]
  for i:=2 to 10 do
 [          ]
  write(s)
 end.
```

(6)

```
program p426;
{利用格里高利公式求π的近似值,直到最后一项的绝对值小于1E-4为止}
 var n,s:integer;
     p:real;
 begin
```

```
    p:=0;
    [          ]
    repeat
     p:=p+s/n;
     [          ]
    until [          ]
     write(p)
   end.
```

4.3　阅读下列程序,写出程序的执行结果。

(1)

```
  program p431;
   var i,s,t:integer;
   begin
    t:=0;s:=0;
    for i:=-5 to 5 do
    begin
     t:=t+1;
     s:=s+t+i
    end;
    writeln('s,t=',s:10,t:10)
   end.
```

(2)

```
  program p432;
   var p,m:integer;
   begin
    p:=20; m:=2;
    repeat
     p:=p-m;
     m:=m+3
    until m>p;
    writeln('m,p=',m:10,p:10)
   end.
```

(3)

```
program p433;
 var a,n:integer;
 begin
  n:=6;a:=0;
  while n>1 do
  begin
   a:=1;
   repeat
    write('*');a:=a+1
   until a>=n;
   writeln;
   n:=n-1
  end
end.
```

(4)

```
program p434;
 var i,j,k,s:integer;
 begin
  s:=0;
  for i:=3 downto 1 do
  begin
   for j:=1 to 3 do
   begin
    k:=0;
    repeat
     k:=k+1;
     s:=s+k
    until k=j
   end;
   s:=s-k-1
  end;
  write('s=',s)
end.
```

(5)

```
program p435;
 var r,c,i:integer;
 begin
  i:=20;
  for r:=1 to 5 do
  begin
   write(' ':i);
   for c:=1 to 2*r-1 do
    write(c:1);
   writeln;
   i:=i-1
  end
 end.
```

4.4 编写下列程序:

(1) 从键盘输入20个整数,分别统计其中正数,负数和零的个数.

(2) 已知数列 $a_0, a_1, a_2, \cdots, a_{20}$ 满足条件

$a_0 = 0$

$a_1 = 1$

$a_2 = 1$

$a_3 = a_0 + 2a_1 + a_2$

$a_4 = a_1 + 2a_2 + a_3$

...

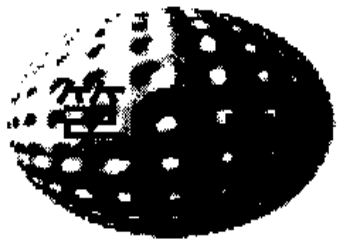
试输出这个数列。

(3) 将一张面值100元的纸币换成若干张面值为1元,2元和5元的纸币,共有多少种换法?要求输出每一种换法。

(4) 用近似公式

$$\frac{\pi^2}{6} \approx \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \cdots + \frac{1}{n^2} + \cdots$$

求π值，直到最后一项的值小于1E-5时为止。



枚举和子界

数据类型丰富、描述数据的能力强，是Pascal语言的优点之一。前面几章介绍的程序处理的都是标准类型(integer, real, char, boolean)的数据。这一章将学习枚举类型数据和子界类型数据的用法。枚举类型和子界类型都属于非标准的简单类型。

5.1 类型定义

如果想在程序中用非标准类型来说明某个变量，那么在进行变量说明之前，首先必须给出这种非标准类型的类型定义。

类型定义的格式为

```
类型名=自定义类型
```

其中，类型名是程序设计者自己选定的一个标识符。

在Pascal语言中程序设计者可以自己定义的 8种类型是：枚举、子界、数组、字符串、集合、记录、指针和文件。在同一个程序中可以定义多个类型，各类型的定义一般集中在以保留字type开始的类型定义部分，每个类型定义的后面跟一个分号，即：

```
类型名1=自定义类型1;
类型名2=自定义类型2;
  …
类型名n=自定义类型n;
```

一个标识符被定义成某种类型的名称之后，在程序的变量说明部分就可以用这个类型名来说明有关的变量：

```
var
  变量名:类型名;
```

有时，也可以将类型定义并入变量说明中，即把

```
type
  类型名=自定义类型;
var
  变量名:类型名;
```

简化成

```
var 变量名:自定义类型;
```

5.2 枚举类型

5.2.1 枚举类型的定义

枚举类型定义的格式为

枚举类型名=(枚举值1,枚举值2, …, 枚举值n)

其中，每个枚举值都是标识符.

[例5.1] 定义两种枚举类型，分别表示一年的四季和一周的每一天。

```
type
  season=(Spring,Summer,Autumn,Winter);
  week=(Sun,Mon,Tue,Wed,Thu,Fri,Sat);
```

在定义枚举类型时，圆括号内枚举值的数目由程序设计者自己决定，每个枚举值必须是标识符，而且不允许重复定义.

[例5.2] 下列定义枚举类型的方式都是错误的.

```
type
  colour=('red','yellow','blue');    {字符串不是标识符，不可以作为枚举值}
  number(1,3,5,7,9);                 {数字不可以作为枚举值}
  week=(Sun, …, Sat);                {每个枚举值必须一一列出，不可以用省略号代替}
  fruits=(apple,orange,pear,pear);   {标识符pear不可以作为两个枚举值}
```

```
drink=(tea,milk,orange);  {标识符orange已作为fruits类型的一个值，
                           不可以再用作drink类型的值}
```

5.2.2 枚举变量的说明

说明枚举变量的方法和说明预定义类型变量的方法完全相同.

[例5.3] 说明一个枚举变量，它可以分别取红、黄、蓝3种枚举值.

方法1：先定义枚举类型，然后说明枚举变量。

```
type
  color=(red,yellow,blue);
var
  p:color;
```

方法2：直接说明枚举变量。

```
var
  p:(red,yollow,blue);
```

两种方法的区别在于：前者，定义了一个枚举类型，将它命名为color，接着用color说明了一个枚举变量p．在程序的其它地方，还可以用color来说明别的变量．后者，定义了一个枚举类型，但没有给它命名．只是在这里用它来说明一个枚举变量p，而不准备在程序的其它地方用它来说明别的变量．

枚举值、枚举类型名和枚举变量名都是标识符，初学者很容易搞混．因此，要记住：我们是用一组枚举值定义枚举类型名，用枚举类型名来说明枚举变量．枚举值和枚举类型名都不可以当作变量来使用．

5.2.3 枚举量的运算

枚举量的运算种类较少：可以将一个枚举值赋值给一个枚举变量；也可以将一个枚举变量赋值给另一个枚举变量．但是，这些赋值必须在同一种枚举类型之间进行．

[例5.4] 若有下面的类型定义、变量说明：

```
type
  color1=(red,yellow,blue);
  color2=(black,white,green,brown);
var
  c1,c2:color1;
```

```
    c3,c4:color2;
```

则下列赋值是允许的：

```
    c2:=red;
    c3:=green;
    c4:=c3
```

但下列赋值则是不允许的：

```
    c1:=white;  { white属于color2类型,c1属于color1类型}
    c2:='red';  { 'red'是字符串，不同于枚举值red}
    c3:=c2      { c2和c3属于不同的枚举类型}
```

还可以对枚举量进行等于（=）、不等于（<>）、小于（<）、不小于（>=）、大于（>）、不大于（<=）这6种关系运算。

在定义枚举类型时，圆括号中每个枚举值都对应一个序号：第一个枚举值的序号为0，第二个枚举值的序号为1，……依次类推。在对枚举量进行关系运算时，比较的依据是两个枚举值所对应的序号的大小。

和赋值运算一样，关系运算也必须在同一种枚举类型之间进行。

[例5.5]　根据例5.4中给出的类型定义、变量说明，允许进行下列关系运算：

```
    red<blue   {结果为true }
    c4>black   {若c4当前值为white、green或brown，则结果为true，否则
                为false }
    c3=c4      {若c3和c4当前值相同，则结果为true，否则为false }
```

但不允许进行下列关系运算：

```
    green>yellow    { green和yellow属于不同的枚举类型}
    red<2           { red是枚举值，而2不是枚举值}
    c2=c3           { c2和c3属于不同的枚举类型}
```

对枚举量只能进行如上所述的赋值和关系运算，不允许进行加、减、乘、除等算术运算。

可用枚举量作参数的标准函数有3个：前趋函数(pred)、后继函数(succ)和序数函数(ord)。

[例5.6]　根据例5.4中给出的类型定义，允许进行下列函数调用：

```
    pred(yellow)    {函数值为red}
    succ (yellow)   {函数值为blue}
    ord(yellow)     {函数值为1}
```

但不允许进行下列函数调用：

pred(red) {red是 color1类型的第一个枚举值,它没有前趋}

succ(blue) {blue是 color1类型的最后一个枚举值,它没有后继}

至于函数调用pred(c1)和succ(c3)是否允许，则取决于变量c1和c3当前的值：如果c1当前值是yellow或blue，则允许调用pred(c1)，否则不允许；如果c3当前值是black、white或green，则允许调用succ(c3)，否则不允许.

5.2.4 枚举量的输入和输出

枚举量不能直接用read语句和write语句进行输入和输出．因此，在程序中出现语句

read(c2)或read(yellow)

write(c2)或write(yellow)

都是错误的.

可以采用类似于输入布尔值的方法输入枚举值.

[例5.7] 根据例5.4中给出的类型定义和变量说明，可用下面的程序段输入枚举变量c2的值：

```
read(i);
case i of
 1: c2:=red;
 2: c2:=yellow;
 3: c2:=blue
end;
```

这里，假设i是经过说明的整型变量．如果想让c2得到枚举值yellow，那么，在执行read(i)这个语句时，就输入整数2.

枚举量的输出可以采用与输入类似的方法：在case语句中根据要输出的枚举变量的值，输出一个相应的字符串.

[例5.8] 根据例5.4中给出的类型定义和变量说明，可用下面的程序段输出枚举变量c2的值：

```
case c2 of
 red    :write('red');
 yellow:write('yellow');
 blue   :write('blue')
end;
```

5.2.5　应用举例

[例5.9]　设有4种水果：苹果、桔子、香蕉和菠萝，现要任取其中3种水果，不能重复，不计先后顺序，请编写程序列出所有可能的取法。

[解题思路]

为增加程序的可读性，定义一个枚举类型fruits，其4个枚举值为apple、orange、banana、pineapple；说明3个枚举变量i、j、k，用它们作为三重循环的3个循环控制变量；令i从apple变化到orange，j从succ(i)变化到banana，k从succ(j)变化到pineapple，对于i,j,k的每一组取值，通过case语句输出相应的3个字符串。

[程序]

```
program ch59;
 type
  fruits=(apple,orange,banana,pineapple);
 var s:integer;
      i,j,k,f:fruits;
 begin
  writeln('可能的取法:');
  for i:=apple to orange do
   for j:=succ(i) to banana do
    for k:=succ(j) to pineapple do
    begin
     for s:=1 to 3 do
     begin
      case s of
       1:f:=i;
       2:f:=j;
       3:f:=k
      end;
      case f of
       apple     :write('apple     ');
       orange    :write('orange    ');
       banana    :write('banana    ');
```

```
            pineapple:write('pineapple')
          end;
        end;
        writeln
      end
  end.
```

[运行结果]

可能的取法:

apple　orange　banana
apple　orange　pineapple
apple　banana　pineapple
orange　banana　pineapple

在这个程序中，由于j的初值为succ(i)，k的初值为succ(j)，因此，循环过程中始终有i<j<k，从而保证输出的3个枚举值互不相同，而且不重复。

注意：i循环的终值是orange,j循环的终值是banana。为什么不能像k循环那样，将i循环和j循环的终值都写成pineapple呢？请读者思考。

5.3　子界类型

5.3.1　子界类型的定义

子界类型定义的格式为

```
子界类型名=下界..上界
```

其中，下界和上界可以是整型、字符型、枚举型的常量，它们必须属于同一种类型(称为基类型)，而且要求下界不大于上界。

[例5.10]　定义3种子界类型，分别表示考试分数、小写英文字母和每周工作日。

```
type
  score=0..100;
  letter='a'..'z';
  week=(Sun,Mon,Tue,Wed,Thu,Fri,Sat);
  workday=Mon..Fri;
```

注意：在定义某个子界类型时，如果基类型是枚举类型，则应首先定义枚举类型，然后再定义子界类型。

[例5.11]　下列定义子界类型的方式都是错误的。

```
type
  score=100..0;              { 下界不可以大于上界}
  salary=0.00..1000.00;      { 基类型不可以是实型}
  workday=Mon..Fri;          { 基类型尚未定义}
  week=(Sun,Mon,Tue,Wed,Thu,Fri,Sat);
```

5.3.2　子界类型的使用

在程序中，每个变量的取值都有一个确定的范围。例如，人的年龄一般不超过150岁，一个月最多只有31天，等等。使用子界类型可以提高程序的可读性，因为从子界类型变量的取值范围很容易猜想到该变量的作用。更重要的是，可以保证程序的正确性。因为在程序执行过程中，当某个子界类型变量的值超出子界规定的范围时，系统会给我们一个出错提示。

对于子界类型的变量可以进行哪些运算？这要看子界类型是怎么定义的，即它的基类型是什么类型。适用于基类型的任何运算都适用于子界类型，只是取值范围受到了限制。

[例5.12]　如果有类型定义和变量说明：

```
type
  score=0..100;
var
  s1,s2:score;
  s:integer;
```

则下面的5个赋值语句中，(1)和(2)是正确的；(3)是错误的；(4)和(5)可能正确也可能不正确：

```
(1) s2:=s1      { s1和s2的取值范围完全一致}
(2) s:=s1       { s1的取值范围只是s取值范围的一部分}
(3) s2:=120     { 120超出了s2的取值范围}
(4) s2:=s1*s1   {只要s1当前的值不大于10就没问题，否则出错}
(5) s2:=s       {只要s当前的值在0至100之间就没问题，否则出错}
```

5.3.3 编译开关命令{$R+}

如果希望像以上所说的那样，在程序运行过程中，当某个子界类型变量的值超出了子界范围时，系统立即给出相应的出错提示，就必须在程序中再加入一条编译开关命令：

{$R+}

它的作用是告诉系统进行范围检查。这条命令通常放在程序最前面。

然而，进行这种范围检查会降低程序的执行速度，并使目标程序代码增长。因此，在实际应用中，只是在调试程序时使用这条命令，一旦程序调试通过，就删除这条命令。

5.3.4 应用举例

[例5.13] 输入每个学生某门课程的成绩，统计并输出该课程全班平均成绩。

[解题思路]

用子界类型的变量mark表示一个学生的成绩，取值范围从0到100。为防止在输入过程中出现误操作，例如，输入一个负数，在这个程序中，使用了编译开关命令{$R+}。

[程序]

```
{$R+}
program ch513;
 type
  score=0..100;
 var
  mark :score;        {个人成绩}
  avge :real;         {平均成绩}
  count:integer;      {全班人数}
 begin
  writeln('请按学号输入学生成绩');
  count:=0;
  avge:=0;
  repeat
   count:=count+1;
```

```
    write(count,'号:');
    read(mark);              {按学号输入一个学生的分数}
    avge:=avge+mark          {累加到全班总分中去}
  until eof;                 {输入结束}
  avge:=avge/count;          {求平均分}
  writeln('平均成绩:',avge:6:2)
end.
```

[运行实例]

```
请按学号输入学生成绩
1号:90
2号:88
3号:93
4号:79^Z                   （注：^Z表示Ctrl-Z键）
平均成绩: 87.50
```

程序倒数第4行中的eof是个标准函数，用于判断键盘输入是否结束。当读到从键盘输入的最后一个数据时，eof函数值为true，否则为false。为了告诉计算机“现在输入的是最后一个数据”，可在输入该数据之后、按下回车键之前按Ctrl-Z键（先按住Ctrl键，再按字母Z）。

在运行这个程序时，如果输入一个小于0或大于100的数据，系统将中断程序的执行，显示出错信息：

```
Error 201 Range check error
```

并将光标停在read(mark)语句处。

[例5.14] 输入一个十进制小数，将它转换成八进制小数，要求取到小数点后24位。

[解题思路]

将十进制小数转换成八进制小数，可采用反复进行“乘8取整”的方法。例如，将十进制小数0.39转换成八进制小数，如果只取到小数点后4位，则进行4次“乘8取整”的操作，即可得到结果0.3075（八进制小数）：

算式	整数部分	小数部分
0.39×8	3	0.12
0.12×8	0	0.96
0.96×8	7	0.48
0.48×8	5	0.44

[程序]

```
program ch514;
 const
  maxlen=24;
 var
  n,p :real;
  i   :1..maxlen;
  d   :0..7;
 begin
  write('待转换的十进制小数是:');
  readln(n);
  write('转换后的八进制小数是:');
  if n<0 then begin
   write('-');          {如果是负数,则先输出一个负号,}
   n:=-n                {然后转换其绝对值}
  end;
  write('0.');
  for i:=1 to maxlen do
  begin
   p:=n*8;              {乘8}
   d:=trunc(p);         {取出整数部分}
   n:=p-d;              {取出小数部分}
   write(d:1)           {输出一个数字}
  end
 end.
```

[运行实例]

```
待转换的十进制小数是:0.39
转换后的八进制小数是:0.307534121727020000000000
```

程序倒数第5行中的trunc是个标准函数，其一般格式为

trunc(实型表达式)

函数值是这个实型表达式值的整数部分．例如，

trunc(6.8)=6

trunc(-6.8)=-6

习 题 五

5.1 填空题。

(1) 用来表示类型定义的保留字是_____。

(2) 用自定义类型来说明一个变量,这个类型必须_____。

(3) 可以用作枚举值的只能是_____。

(4) 在枚举类型的定义中,第一个枚举值对应的序号是_____。

(5) 对枚举类型的数据只能进行_____和_____运算。

(6) 在对枚举量进行关系运算时,比较的依据是_____。

(7) 在定义子界类型时,下界和上界只能是_____、_____、_____的常量。

(8) 在定义子界类型时,下界和上界所属的类型称为_____。

(9) 对子界类型的数据可以进行哪些运算取决于_____。

(10) 编译开关命令{$R+}的作用是_____。

5.2 判断下列类型定义是否正确.

(1) type sex=(male,female)

(2) type ch=('A','B','E','F')

(3) type b=(false,true)

(4) type L=(while,repeat,for)

(5) type bc=(black,white,blue);fc=(red,yellow,blue,brown)

(6) type a=(d,b,p)

(7) type n=-0.5..11.5

(8) type s=1..2*10

(9) type a='a'..'u'

(10) type c=min..max

5.3 设有类型定义如下:

```
type
  color1=(red,yellow,blue);
  color2=(black,white,green,brown)
```

请写出下列表达式的值:

(1) succ(white)

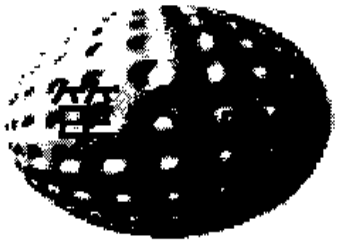
(2) succ(succ(red))

(3) pred(green)

(4) ord(brown)+1

(5) ord(pred(yellow))>ord(succ(black))

5.4　一周7天用sun, mon, tue, wed, thu, fri, sat 表示，编程序输入一个数字(0~6)，要求打印出昨天，今天和明天各是星期几。



数组和字符串

变量是用来存放数据的。在前几章的程序中所使用的变量有一个共同的特点：在同一时刻每个变量只能存放一个数据。这种变量称为**简单变量**。

在许多情况下，只用简单变量要编写出一个正确、高效的程序是非常困难的。请看下面这个例子。

[引例] 先从键盘输入10个整数，然后按与输入时相反的顺序输出这10个整数。

若使用简单变量，则可能写出如下程序段：

```
read (s1,s2,s3,s4,s5,s6,s7,s8,s9,s10);
write(s10,s9,s8,s7,s6,s5,s4,s3,s2,s1)
```

虽然达到了目的，但是太繁琐。

当需要处理一批类型相同的数据时，通常采用**数组变量**。对于引例所提出的问题，使用数组变量，程序段是非常简洁的：

```
for i:=1 to 10 do read(s[i]);
for i:=10 downto 1 do write(s[i])
```

在这个程序段中，s称为数组变量，用于存放10个整数；s[i]称为数组元素，用于存放第i个整数，方括号内的i称为**下标**。输入时，i从1变化到10，通过read(s[i])语句将从键盘读入的10个整数依次存入s[1],s[2],…, s[10]这10个数组元素中；输出时，i从10变化到1，通过write(s[i])语句依次输出存放在s[10],s[9], …, s[1]中的值。

在Pascal语言中，数组是一种用得最多的自定义类型，它由一批类型相同的数据构成，这些类型相同的数据称为数组元素。根据数组元素排列方式的不同，数组分为一维数组、二维数组、三维数组，等等。本章重点介绍一维数组和二维数组。

6.1　一维数组

一维数组中的数据排成一行，对应于数学中的一个向量：

$(a_1, a_2, a_3, \cdots, a_n)$

6.1.1　定义一维数组

一维数组类型定义的格式为

数组类型名=array[下标类型] of 元素类型

其中，array和of是保留字；下标类型只能是子界类型或枚举类型（严格地说，还可以是布尔类型，但一般不用），它规定了下标的取值范围；数组元素类型可以是除文件以外的任何数据类型。

[例6.1]　定义一维数组类型，并用于说明引例中的数组变量s。

```
type
 index=1..10;
 arraytype=array[index] of integer;
var s:arraytype;
```

或者省略子界类型定义：

```
type
 arraytype=array[1..10]of integer;
var s:arraytype;
```

或者将类型定义并入变量说明中：

```
var s:array[1..10]of integer;
```

6.1.2　使用一维数组

在程序执行部分，使用数组的方法有两种：一种是使用数组变量；另一种是使用数组元素．通常是采用第二种方法。

一维数组元素的格式为

数组变量名[下标表达式]

其中，下标表达式的类型要和说明这个数组时用的下标类型相同，下标表

达式的值必须限定在下标类型的取值范围内。

数组元素可以像同类型的简单变量那样使用：可以出现在相应的表达式中或赋值号的左边；如果是整数、字符、实数等类型，还可以用read、write语句输入输出。总之，凡是简单变量可以出现的地方一般都可以出现相同类型的数组元素。但是，不可以用作for语句的循环控制变量。

给数组变量赋值可以采用两种方法：

①对每个数组元素逐一赋值；

②对数组变量整体赋值。

[例6.2]　设有如下类型定义和变量说明：

```
type
  mark=0..100;
  cours=(Chinese,Politics,English);
  score=array[cours]of mark;
var
  s1,s2 :score;
  i     :cours;
```

则语句

```
s2:=s1
```

和语句

```
for i:=Chinese to English do s2[i]:=s1[i]
```

具有相同的功能，都是将s1的3个元素赋值给s2的3个对应元素。

显然，第一种方法要简单些。但要注意：只有相同类型的数组才可以进行整体赋值。所谓“相同类型”，在这里指什么呢？简单地说，是要求两个数组是用同一个数组类型名或数组类型定义来说明的（有相同的元素类型，有同样多的元素，下标类型也相同）。

对数组变量除了进行这种整体赋值外，不能再进行其它操作。

[例6.3]　根据例6.2中给出的类型定义和变量说明，下面这些语句都是错误的：

```
s1:=s1+s2;  { 企图把s2的每个元素值叠加到s1的对应元素上 }
s1:=2*s1;   { 企图使s1的每个元素值乘上系数2          }
if s1=s2 then write ('ok'); { 企图比较两个数组是否相等 }
read(s1);   { 企图用一个read语句输入s1的所有元素   }
write(s1)   { 企图用一个write语句输出s1的所有元素 }
```

6.1.3 定义一维数组常量

有时会碰到这样的问题，每次执行一个程序时都需要给其中的某个数组变量赋相同的初始值（所谓相同的初始值，就是给每个元素每次赋相同的初始值，而各元素的初始值可以不相同）。如果数组中元素很多，每次从键盘输入这些初始值是很乏味的。解决这个问题的一个较好的方法是：先将这组初始值定义成一个数组常量，然后将这个数组常量整体赋值给需要赋初始值的数组变量。

一维数组常量定义的格式为

```
数组常量名:数组类型=(初值1,初值2,…,初值n)
```

其中，圆括号内初值的个数取决于一维数组类型的下标类型，初值的类型要与数组元素类型相同。

[例6.4] 用数组常量p0给数组变量p赋初始值（10个实数）。

```
type
  index=1..10;
  coefficient=array[index]of real;
const
  p0:coefficient=( 0.25,0.00,0.30,0.20,0.15,
                   0.10,0.31,0.22,1.00,0.16 );
var
  p:coefficient;
  …
  p:=p0;   { 在执行部分对数组进行整体赋值 }
  …
```

由于在定义常量p0时，要用到类型名coefficient，所以应该先定义数组类型，后定义数组常量。

[例6.5] 用数组常量c0给数组变量c赋初始值(16个字符)。

```
type
  index=1..16;
  hexadecimal=array[index]of char;
const
  c0:hexadecimal=('0','1','2','3','4','5','6','7','8','9', 'A','B','C','D','E','F');
```

```
var
  c:hexadecimal;
 …
 c:=c0;    { 在执行部分对数组进行整体赋值 }
 …
```

当一维数组的元素类型是字符类型时，数组常量的定义形式可以简化．对本例来说，c0也可以这样定义：

```
const
  c0:hexadecimal='0123456789ABCDEF';
```

数组常量除了可用来给同类型的数组变量赋初始值外，还可以通过下标使用其中某个元素．例如，c0[10]是数组常量c0的第10个元素，其值为'A'．

在形式上，数组常量和数组变量、数组常量的元素和数组变量的元素没有区别．但实质上是有区别的：在程序执行过程中，常量的值始终保持不变，而变量的值是可以改变的．因此，试图通过赋值来改变数组常量的做法是错误的．

6.1.4 应用举例

[例6.6] 从键盘读入一个以句号结尾的英文句子，统计并输出句中各字母出现的次数．假设句中没有大写字母．

[解题思路]

引入一个下标类型为子界类型'a'..'z'的一维数组变量count，用于存放句中各字母出现的次数．当读入一个字符ch后，若ch是字母，则令count[ch]加1．当读入句号后，将数组count中不为0的元素列表输出．

[程序]

```
program ch66;
 var
  ch:char;
  count:array['a'..'z']of integer;
  k:0..4;
 begin
  {初始化}
  for ch:='a' to 'z' do
   count[ch]:=0;
```

```
  {输入}
  writeln('请输入一个以句号结尾的英文句子');
  repeat
   read(ch);                          {读入一个字符}
   if (ch>='a')and(ch<='z')           {如果它是字母}
   then count[ch]:=count[ch]+1        {该字母出现次数加1}
  until ch='.';
  {输出}
  writeln('句中各字母出现次数如下:');
  k:=0;                              {列计数}
  for ch:='a' to 'z' do
   if count[ch]>0 then               {如果句中出现过该字母}
   begin                             {输出该字母及出现次数}
    write('    ',ch,': ',count[ch]:2,'次');
    k:=(k+1)mod 4;                   {每行只输出4列}
    if k=0 then writeln
   end
 end.
```

[运行实例]

请输入一个以句号结尾的英文句子

the kings of ancient egypt planned strong tombs to keep their bodies safe after death and to hold their treasures.

句中各字母出现次数如下:

a: 7次	b: 2次	c: 1次	d: 5次
e: 14次	f: 3次	g: 3次	h: 5次
i: 5次	k: 2次	l: 2次	m: 1次
n: 7次	o: 7次	p: 3次	r: 6次
s: 7次	t: 12次	u: 1次	y: 1次

[例6.7]　从键盘读入n个整数，进行从小到大排序，然后输出。

[解题思路]

先将n个整数输入到数组元素a[1],a[2], …, a[n]中，然后依次进行下列操作：从a[1]~a[n]中选出值最小的元素，如果它不是a[1]，就将它与a[1]交换；从a[2]~a[n]中选出值最小的元素，如果它不是a[2]，就将它与a[2]交换；……最后，从a[n-1]和a[n]中选出值最小的元素，如果它不是a[n-1]，

就将它与a[n-1]交换．这种排序方法称为选择排序。

[程序]

```
program ch67;
 const n=10;                          {假设对10个整数进行排序}
 var
  a:array[1..n]of integer;
  i,j,k,t:integer;
 begin
 {输入}
  writeln('输入',n:3,'个整数:');
  for i:=1 to n do read(a[i]);
  {排序}
  for i:=1 to n-1 do
  begin
  {从a[i]~a[n]中选出一个最小值,k指出最小值在数组中的下标}
   k:=i;                              {先假设a[i]是最小值}
   for j:=i+1 to n do                 {然后检查a[i+1]~a[n]}
    if a[j]<a[k] then k:=j;           {如果找到一个更小的,则让k指向它}
   if k>i                             {循环结束后,如果最小值不是a[i]}
   then begin                         {则交换a[i]和a[k]的值}
    t:=a[k];a[k]:=a[i];a[i]:=t
   end
  end;
  {输出}
  writeln('排序结果:');
  for i:=1 to n do write(a[i],' ')
 end.
```

[运行实例]

```
输入10个整数:
519  49   12   557  552  66   15   503  608  515
排序结果:
12   15   49   66   503  515  519  552  557  608
```

[例6.8]　从键盘读入n个整数

$a_1, a_2, \cdots, a_k, a_{k+1}, \cdots, a_n$

对其进行平移处理，按

$a_{k+1}, a_{k+2}, \cdots, a_n, a_1, a_2, \cdots, a_k$

的顺序输出，其中k是小于n的常数。

[解题思路]

将n个整数输入到整型数组a中，重复k次下列步骤，完成平移操作：

① 将a[1]中的值保存到临时单元t中；

② 将a[2]~a[n]中的值顺序左移一个位置；

③ 将临时单元t中的值存到a[n]中。

步骤②是个循环过程，操作时要先把a[2]中的值移到a[1]中，然后把a[3]中的值移到a[2]中，……最后把a[n]中的值移到a[n-1]中，不可倒过来，否则会把a[1]~a[n]中的内容都变成a[n]中的值。

[程序]

```
program ch68;
 const n=10;
       k=3;
 var   a:array[1..n]of integer;
  i,j,t:integer;
 begin
  { 输入 }
  writeln('输入',n,'个整数:');
  for i:=1 to n do read(a[i]);
  { 平移k次 }
  for i:=1 to k do
  begin
   t:=a[1];        {保存a[1]}
   for j:=2 to n do a[j-1]:=a[j]; {顺序左移}
   a[n]:=t     {a[1]移入a[n]}
  end;
  { 输出 }
  writeln('平移',k,'次后:');
  for j:=1 to n do write(a[j]:4)
 end.
```

[运行实例]

输入10个整数:

12　15　49　66　503　515　519　552　557　608

平移3位后:

66　503　515　519　552　557　608　12　15　49

[例6.9]　精确计算n的阶乘n!（n大于7小于50）

[解题思路]

因为8!=40320，已超出integer类型的表示范围，所以不能用integer类型来表示n!(也不能用实数类型，因为实数运算有误差).

可以用一个一维数组来表示这个大数，数组中每个元素只存放一个数字。因为50!小于10的100次方，所以数组下标类型用子界1..100。下图所示的是大数3628800在数组中的存储方法。

1	2		94	95	96	97	98	99	100
			3	6	2	8	8	0	0

假设某个大数的各位从高位到低位已存放在a[k]~a[100]中，求整数i与这个大数的乘积可按以下步骤进行：

① 令x的初始值为0；　　{x表示来自低位的进位}

② j从100到k，重复执行下列操作：

```
x:=x+a[j]*i;
a[j]:=x mod 10;
x:=x div 10          {产生新的进位}
```

③ 重复执行下列操作，直到x=0为止：

```
k:=k-1;
a[k]:=x mod 10;
x:=x div 10
```

为了计算n!，只须令k=100，a[k]=1，然后通过一个循环，用i作为循环控制变量，从2到n，执行上述3个步骤即可.

[程序]

```
program ch69;
 const max=100;                {数组最大下标}
       n=20;                   {计算20!}
 var a:array[1..max]of 0..9;
     i,j,k,x:integer;
 begin
  k:=max;a[k]:=1;
```

```
  for i:=2 to n do
  {每循环一次将保存在a[k]~a[max]中的大数乘上一个i}
   begin
    x:=0;
    for j:=max downto k do
     begin
      x:=x+a[j]*i;                    {本位积加上来自低位的进位}
      a[j]:=x mod 10;                 {取出个位数字}
      x:=x div 10                     {产生新的进位}
     end;
   {将最后一次进位的各位数字取出并填入数组}
    while x>0 do
     begin
      k:=k-1;
      a[k]:=x mod 10;
      x:=x div 10
     end
   end;
  writeln;
  for i:=k to max do write(a[i]:1)
 end.
```

[计算结果]

2432902008176640000

[例6.10] 制作一个生日蛋糕的原料配方及每种原料的价格如下表所示：

原 料	水果	黄油	糖	面粉	鸡蛋	白兰地
份量(克)	100	400	400	50	250	150
价格(元/克)	0.25	0.30	0.20	0.05	0.20	0.15

写一个程序，计算制作n个蛋糕的原料总价格。

[解题思路]

由于原料配方及价格是固定的，因此用数组常量较合适。

[程序]

```
program ch610;
```

```
type
  compon=(fruit,cheese,suger,flour,egg,wine);
  col=array[compon]of real;
const
  wt:col=(100, 400, 400, 50, 250, 150);
  p :col=(0.25,0.30,0.20,0.05,0.20,0.15);
var
  i :compon;
  cp:real;        {原料总价格}
  n :integer;     {蛋糕数量}
begin
  write('蛋糕数量n=');
  readln(n);
  cp:=0;
  for i:=fruit to wine do
    cp:=cp+wt[i]*p[i];
  write('原料总价格为:',cp*n:8:2,'元')
end.
```

[运行实例]

蛋糕数量n=8

原料总价格为: 2400.00元

6.2　二维数组

二维数组中的元素被排成若干行、若干列，每一行的元素个数相同，每一列的元素个数也相同．它对应于数学中的矩阵：

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & & \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

6.2.1　定义二维数组

二维数组类型定义的格式为

```
数组类型名=array[下标类型1,下标类型2] of 元素类型
```

其中，下标类型1规定行下标的取值范围，下标类型2规定列下标的取值范围，下标类型1和下标类型2都只能是子界型或枚举型(严格地说，还可以是布尔型，但一般不用)；元素类型可以是除文件以外的任何数据类型。

[例6.11]　用二维数组存放某班30名学生数学、物理、化学和生物4门课程的考试成绩。

```
type
  course=(Mathematics,Physics,Chemistry,Biology);
  number=1.. 30;
  mark=0..100;
  table=array[number,course] of mark;
var
  s:table;
```

也可以像说明一维数组变量那样，把二维数组类型定义并入变量说明中：

```
type
  course=(Mathematics,Physics,Chemistry,Biology);
  number=1..30;
  mark=0..100;
var
  s:array[number,course] of mark;
```

此例中，下标类型1是子界型，表示学生人数，取值范围从1到30；下标类型2是枚举型，表示课程，取值范围从Mathematics到Biology。二维数组变量s有30行、4列，共120个元素，其排列方式为：

```
s[1,Mathematics],s[1,Physics],s[1,Chemistry],s[1,Biology]
s[2,Mathematics],s[2,Physics],s[2,Chemistry],s[2,Biology]
…
s[30,Mathematics],s[30,Physics],s[30,Chemistry],s[30,Biology]
```

6.2.2　使用二维数组

与一维数组一样，在程序执行部分一般只使用数组元素，很少使用数组变量。

二维数组元素的格式为

数组变量名[下标表达式1,下标表达式2]

二维数组元素有两个下标，下标表达式1是元素的行下标，下标表达式2是元素的列下标，它们的值必须分别限定在下标类型1和下标类型2的取值范围之内。

二维数组元素也可以像同类型的简单变量那样使用。

除了整体赋值外，对二维数组变量不能进行其它操作。对二维数组变量进行整体赋值时，要求两个数组变量属于同一种二维数组类型。

6.2.3　定义二维数组常量

二维数组常量定义的格式为

```
二维数组常量名:二维数组类型=((初值11,初值12,…,初值1n),
                              (初值21,初值22,…,初值2n),
                                    …
                              (初值m1,初值m2,…,初值mn))
```

每一行的初始值用圆括号括起来，各行之间用逗号分隔，在每一行中，各初始值也用逗号分隔。

[例6.12]　将矩阵U0赋值给数组变量u.

$$U0=\begin{bmatrix}1&0&0&1&0\\0&0&1&0&1\\1&1&0&1&0\\0&0&1&0&1\end{bmatrix}$$

```
const m=4; n=5;
type
  matrix=array[1..m,1..n] of integer;
const
```

```
  U0: matrix=((1,0,0,1,0),(0,0,1,0,1),
              (1,1,0,1,0),(0,0,1,0,1));
var
  u:matrix;
  …
  u:=U0;    {在执行部分中出现}
```

6.2.4　应用举例

[例6.13]　从键盘读入n个学生数学、物理、化学、生物这4门课程的考试成绩，计算每门课程n个学生的平均分数和每个学生4门课程的平均分数，并以表格形式输出。

[解题思路]

如果题目不要求以表格形式输出结果，则可以不用数组：

```
n:=0;
s1:=0; s2:=0;
s3:=0; s4:=0;
repeat
 readln(x1,x2,x3,x4);  { 读入一个学生4门课程的分数 }
 writeln((x1+x2+x3+x4)/4);  { 输出他的平均分数 }
 s1:=s1+x1; s2:=s2+x2;  { 把他的分数累加到全班总分中去 }
 s3:=s3+x3; s4:=s4+x4;
 n:=n+1 { 学生人数加1 }
until eof;
writeln(s1/n,s2/n,s3/n,s4/n)  { 输出各科全班平均分 }
```

现要求以表格形式输出，故必须将输入和输出的过程分开，把全班同学的成绩都输入以后再进行输出。

用n行4列的二维数组s表示成绩表，s的行下标从0到n，其中，第0行存放各科全班平均分，第i行（i>0）存放第i个学生的各科成绩。每个学生4门课程的平均分数用一维数组ave表示。

[程序]

```
program ch613;
 const
  maxn=50;    {假定一个班的学生人数不超过50人}
```

```
type
 number=0..maxn;
 course=(math,phys,chem,biol);
var
 s:array[number,course]of real;
 ave:array[number]of real;
 i,n:number;
 k:course;
begin
 { 初始化 }
 n:=0;  {n表示序号}
 for k:=math to biol do
  s[0,k]:=0;
 { 输入 }
 writeln('请按顺序输入数学,物理,化学,生物成绩');
 repeat
  n:=n+1;
  ave[n]:=0;
  write(n,'号:');
  {输入某个学生各科分数}
  for k:=math to biol do
   begin
    read(s[n,k]);
    ave[n]:=ave[n]+s[n,k]; {计算该学生的总分}
    s[0,k]:=s[0,k]+s[n,k]  {计算该课程全班总分}
   end;
  ave[n]:=ave[n]/4;          {计算该学生的平均分}
 until eof;     {用按Ctrl-Z控制输入结束}
 { 输出 }
 writeln;
 writeln('              成  绩  单              ');
 writeln('序号  数学  物理  化学  生物  平均');
 for i:=1 to n do
  begin
```

```
    write(i:3,' ');              {输出一个序号}
    for k:=math to biol do  {输出各科分数}
     write(s[i,k]:6:1);
    writeln(ave[i]:6:1)       {输出该学生的平均分并换行}
   end;
  write('平均');                 {输出每门课程全班的平均分}
  for k:=math to biol do write(s[0,k]/n:6:1)
 end.
```

[运行实例]

请按顺序输入数学、物理、化学和生物的成绩

1号: 80　87　85　90

2号: 83　67　83　86

3号: 86　89　90　86

4号: 90　88　87　92

5号: 78　81　83　86^Z　（注：^Z表示Ctrl-Z键）

成　绩　单

序号	数学	物理	化学	生物	平均
1	80.0	87.0	85.0	90.0	85.5
2	83.0	67.0	83.0	86.0	79.7
3	86.0	89.0	90.0	86.0	87.7
4	90.0	88.0	87.0	92.0	89.2
5	78.0	81.0	83.0	86.0	82.0
平均	83.4	82.4	85.6	88.0	

[例6.14]　输入一个m行m列的矩阵，求它的转置，即对所有的i、j，将A[i,j]与A[j,i]交换。

[解题思路]

下面这种做法是错误的：

```
for i:=1 to m do
 for j:=1 to m do
  begin
   t:=a[i,j];a[i,j]:=a[j,i];a[j,i]:=t
  end;
```

这样做的结果是，除主对角线上的元素外，每个元素被交换了两次，因此交换前后数组内容不变。当m＝3时，这种错误的处理过程如图6.1所示。

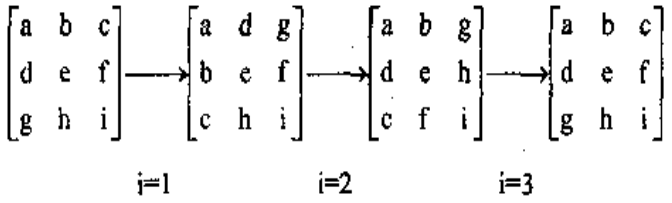


图6.1　错误的处理过程

正确的做法是，把内循环控制变量j的终值由m改为i-1，使每个元素只被交换一次。

[程序]

```
program ch614;
 const m=4;
 type index=1..m;
       matrix=array[1..m,1..m]of integer;
 var a:matrix;
     i,j:index;
     t:integer;
 begin
  { 输入 }
  writeln('输入',m:2,'阶矩阵:');
  for i:=1 to m do
   for j:=1 to m do read(a[i,j]);
  { 计算 }
  for i:=1 to m do
   for j:=1 to i-1 do
    begin
     t:=a[i,j];a[i,j]:=a[j,i];a[j,i]:=t
    end;
  { 输出 }
  writeln;
  writeln('该矩阵的转置为:');
  for i:=1 to m do
   begin
    for j:=1 to m do write(a[i,j]:4);
    writeln
   end
```

```
 end.
```

[运行实例]

```
输入4阶矩阵:
11 12 13 14
21 22 23 24
31 32 33 34
41 42 43 44
该矩阵的转置为:
11 21 31 41
12 22 32 42
13 23 33 43
14 24 34 44
```

[例6.15] 输入一个m行w列的矩阵A和一个w行n列的矩阵B，计算并输出A和B的乘积C。

[解题思路]

A矩阵和B矩阵相乘，要求A矩阵的列数等于B矩阵的行数，本例满足此要求．乘积C矩阵有m行n列，其中第i行第j列上的元素用下面的公式计算：

C[i,j]=A[i,1]*B[1,j]+A[i,2]*B[2,j]+…+A[i,w]*B[w,j]

在程序中可用一个for循环实现：

```
s:=0;
for k:=1 to w do s:=s+a[i,k]*b[k,j];
c[i,j]:=s
```

计算整个C矩阵，可采用二重循环结构：

```
for i:=1 to m do
 for j:=1 to n do
  {计算C[i, j]}
  begin
   s:=0;
   for k:=1 to w do s:=s+a[i,k]*b[k,j];
   c[i,j]:=s
  end
```

[程序]

```
program ch615;
 const m=4;w=2;n=3;
```

```
var
 a:array[1..m,1..w]of integer;
 b:array[1..w,1..n]of integer;
 c:array[1..m,1..n]of integer;
 i,j,k,s:integer;
begin
 { 输入 }
 writeln('输入',m:2,'行',w:2,'列A矩阵:');
 for i:=1 to m do
  for j:=1 to w do read(a[i,j]);
 writeln('输入',w:2,'行',n:2,'列B矩阵:');
 for i:=1 to w do
  for j:=1 to n do read(b[i,j]);
 { 计算 }
 for i:=1 to m do
  for j:=1 to n do
   {计算C[i, j]}
   begin
    s:=0;
    for k:=1 to w do
     s:=s+a[i,k]*b[k,j];
    c[i,j]:=s
   end;
 { 输出 }
 writeln;
 writeln('乘积C矩阵为:');
 for i:=1 to m do
  begin
   for j:=1 to n do write(c[i,j]:4);
   writeln
  end
end.
```

[运行实例]

输入4行2列A矩阵:

```
1  2
8  3
4  5
9  6
输入2行3列B矩阵:
-1   -5   -2
 2    1    4
乘积C矩阵为:
 3   -3    6
-2  -37   -4
 6  -15   12
 3  -39    6
```

[例6.16]　输入由m×n个互不相同的整数构成的m行n列矩阵a，找出其中在行上最大在列上最小的那个元素（这样的元素称为鞍点）．如果找到了那个元素，则输出该元素的行号和列号；如果没有这样的元素，则输出相应的信息。

[解题思路]

确定第i行上最大值的列下标v；检查第v列上的每个元素，如果没有比a[i,v]更小的元素，则说明a[i,v]就是鞍点，否则i加1(看下一行)．这是一个循环过程，i从1变化到m．但由于找到鞍点后可以提前结束循环，所以不宜用for语句．程序中，一开始给布尔变量found赋初值false，一旦找到了鞍点就将它的值改为true(用来作为退出循环的条件之一)；如果循环结束后found值依然为false，则说明不存在鞍点．

[程序]

```
program ch616;
 const m=4;n=3;
 var
  a:array[1..m,1..n]of integer;
  i,j,u,v:integer;
  found:boolean;
 begin
  { 输入 }
  writeln('输入',m:2,'行',n:2,'列矩阵:');
  for i:=1 to m do
```

```
    for j:=1 to n do read(a[i,j]);
  found:=false;
  i:=1;  {从第1行开始检查}
  repeat
   { 确定第i行上最大值的列下标v }
   v:=1;
   for j:=2 to n do
    if a[i,j]>a[i,v] then v:=j;
   { 检查第v列上有没有比a[i,v]更小的元素 }
   u:=0;
   repeat u:=u+1
   until (u>m)or(a[u,v]<a[i,v]);
   if u>m { 第v列上没有比a[i,v]更小的元素 }
   then fpund:=true { 找到了 }
   else i:=i+1    { 检查下一行 }
  until found or (i>m);
  { 输出 }
  writeln;
  if found
  then write('鞍点在第', i:2, '行第', v:2, '列。')
  else write('没有鞍点。')
 end.
```

[运行实例]

```
输入4行3列矩阵:
95  77  89
24  65  53
63  87  71
11  29  51
鞍点在第 4行第 3列。

输入4行3列矩阵:
95  77  89
24  65  51
63  87  71
```

11 29 53

没有鞍点。

[例6.17] 编写一个根据n的值显示螺旋式数字方阵的程序．方阵中数字的排列规律如下：

n=3 时:

1 2 3
8 9 4
7 6 5

n=4 时:

1 2 3 4
12 13 14 5
11 16 15 6
10 9 8 7

n=5时:

1 2 3 4 5
16 17 18 19 6
15 24 25 20 7
14 23 22 21 8
13 12 11 10 9

[解题思路]

方阵中有n×n个数，可以用n行n列的二维数组来存放这些数，待n×n个数全部产生并存入了这个二维数组后，输出这个二维数组．

用4个整型变量left、right、up、down表示方阵的左、右、上、下边界，其初始值分别为1、n、1、n。

重复执行下列操作,直到n×n个元素都填满：

① 先从左往右填up行，然后up值加1；

② 先从上往下填right列，然后right值减1；

③ 先从右往左填down行，然后down值减1；

④ 先从下往上填left列，然后left值加1．

[程序]

```
program ch617;
 const maxn=20;
 var a:array[1..maxn,1..maxn]of integer;
     i,j,s,left,right,up,down:integer;
     n:integer;
```

```
begin
 write('请输入n值(不能大于',maxn:2,'):');
 readln(n);
 left:=1;right:=n;
 up:=1;down:=n;
 s:=0;  {s是需要填入的数字}
 i:=up;      {准备填up行}
 while s<n*n do
  begin
   {从左往右填up行}
   for j:=left to right do
    begin
     s:=s+1;a[i,j]:=s
    end;
   up:=up+1;
   j:=right;{准备填right列}
   {从上往下填right列}
   for i:=up to down do
    begin
     s:=s+1;a[i,j]:=s
    end;
   right:=right-1;
   i:=down; {准备填down行}
   {从右往左填down行}
   for j:=right downto left do
    begin
     s:=s+1;a[i,j]:=s
    end;
   down:=down-1;
   j:=left;{准备填left列}
   {从下往上填left列}
   for i:=down downto up do
    begin
     s:=s+1;a[i,j]:=s
```

```
      end;
     left:=left+1
     i:=up  {准备填up行}
    end;
   {输出}
   writeln;
   for i:=1 to n do
    begin
     for j:=1 to n do write(a[i,j]:4);
     writeln
    end
  end.
```

[运行实例]

```
请输入n值(不大于20):5
 1   2   3   4   5
16  17  18  19   6
15  24  25  20   7
14  23  22  21   8
13  12  11  10   9
```

[例6.18] 单人纸牌游戏按如下规则进行：把拿掉了大王和小王的52张扑克牌分成13堆，从左往右摆开，每堆4张；翻开第1堆最上面的一张牌，如果它是i（1≤i≤13，其中1表示A，11表示J，12表示Q，13表示K），就把它放到第i堆的下面，此时，若第i堆中的牌已全部翻开，则游戏结束，否则接着翻开第i堆最上面的一张牌，游戏继续进行；当游戏结束时，统计已翻开牌的数目，若已翻开牌的数目超过48张，则认为本次游戏成功，否则就是不成功.

请编写模拟这个游戏的程序.

[解题思路]

实际的扑克牌除了点数外还有花色(红桃、方块、梅花、黑桃)，从本例要求看，在这个程序中，显然不必考虑花色.因此可以用4行13列的二维数组p表示这副扑克牌，每列表示题中所说的一堆.

用随机函数产生这副扑克牌.

给j置初值1，当p[1,j]>0时重复下列操作：

① 把p[1,j]赋值给k（表示翻开第j堆最上面的那张牌）；

② 已翻开牌数n加1；

③ 将p[2,j]~p[4,j]中的值顺序上移一个位置；

④ 把p[4,j]置成0（以后插入第j堆最下面的一定是已翻开的牌，用0表示它）；

⑤ 把k赋值给j（准备翻下一堆）。

在TURBO Pascal中，标准函数random用于产生随机数，它有两种格式。

格式1：

random

其功能是，产生[0,1]之间的一个随机实数。

格式2：

random(x)

其中，x是整型表达式。其功能是，产生[0,x]之间的一个随机整数。

在第一次执行random函数之前，首先要执行randomize语句。randomize语句的功能是给random函数初始化。

[程序]

```
program ch618;
 var p:array[1..4,1..13]of integer;
     c:array[1..13]of integer;
     i,j,k,n:integer;
 begin
  { 产生一副扑克牌 }
  for j:=1 to 13 do
   c[j]:=4; { 准备4张点数为j的牌 }
  { 随机发牌 }
  randomize;
  i:=4;j:=1;
  repeat
   k:=random(13) mod 13+1; { 随机产生k值 }
   if c[k]>0 { 点数为k的牌还没发完 }
   then begin
          p[i,j]:=k; { 把点数为k的牌放到第j堆的上面 }
          c[k]:=c[k]-1;
          j:=j+1;
          if j>13  { 每堆都已发了一张 }
```

```
            then begin
                   i:=i-1; j:=1 { 再从第一堆开始发牌 }
                 end
          end
until i=0;
{ 输出这副扑克牌 }
writeln;
writeln('初局:');
for i:=1 to 4 do
 begin
  for j:=1 to 13 do write(p[i,j]:5);
  writeln
 end;
{ 游戏开始 }
j:=1;
n:=0;
while p[1,j]>0 do
 begin
  k:=p[1,j]; { 翻开第j堆最上面的那张牌 }
  n:=n+1;     { 计数 }
  for i:=2 to 4 do p[i-1,j]:=p[i,j];
  p[4,j]:=0;
  j:=k
 end;
{ 游戏结束 }
if n>48
then writeln('翻开',n:2,'张(成功).')
else writeln('翻开',n:2,'张(失败).');
writeln('终局:');
for i:=1 to 4 do
 begin
  for j:=1 to 13 do write(p[i,j]:5);
  writeln
 end
```

```
end.
```

[运行实例]

初局:

```
 2  7  6  5 13  7  4  4  6 13 10 10 10
12 12  9  7  2  4  3  8  8  8  3  3  4
 1 11  8  9  6 13  3 11 12 13 10  5 12
11  9 11  5  1  2  1  5  1  6  2  7  9
```

翻开 52张(成功).

终局:

```
 0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0
```

初局:

```
 8  3  3  4 13  2 11  8  1 10 11 11 13
 2  4  5  1  6  3  6 13  6  1  8  1  5
 4  4 12  5 12 12  9  7  2 10  9  3  9
13  2  8  7 10  5  6 12  7  7 11  9 10
```

翻开 34张(失败).

终局:

```
 0  4  8  7 10 12  6 12  2 10 11  3  0
 0  2  0  0  0  5  9  0  7  7  0  9  0
 0  0  0  0  0  0  6  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0
```

[例6.19] 制作大、中、小3种蛋糕的原料配方及每种原料的价格如下表所示:

原　料	水果	黄油	糖	面粉	鸡蛋	白兰地
份量(克)	100	400	400	50	250	150
	75	300	300	25	200	100
	50	200	200	20	150	50
价格(元/克)	0.25	0.30	0.20	0.05	0.20	0.15

写一个程序，计算制作每种蛋糕的原料价格.

[解题思路]

3种蛋糕的原料配方用二维数组常量表示，价格用一维数组常量表示.

[程序]

```
program ch619;
 type
  kinds=(L,M,S);      {表示三种蛋糕的规格}
  compon=(fruit,cheese,suger,flour,egg,wine);
  col=array[kinds,compon]of real;
 const
  wt:col=((100,400,400, 50,250,150),      {大型蛋糕的配方}
          ( 75,300,300, 25,200,100),      {中型蛋糕的配方}
          ( 50,200,200, 20,150, 50));     {小型蛋糕的配方}
  p: array[compon]of real
    =(0.25,0.30,0.20,0.05,0.20,0.15);     {各种原料的价格}
 var
  i:compon;j:kinds;
  cp:array[kinds]of real;
 begin
  writeln('规格   原料价格(元)');
  for j:=L to S do
   begin
    cp[j]:=0;
    for i:=fruit to wine do
     cp[j]:=cp[j]+wt[j,i]*p[i];
    case j of
     L  : write(' 大');
     M  : write(' 中');
     S  : write(' 小')
    end;
    writeln(cp[j]:10:2)
   end
 end.
```

[运行结果]

```
规格      原料价格（元）
 大       300.00
```

中　　225.00
小　　151.00

6.3　多维数组

下标多于一个的数组称为多维数组．前面介绍的二维数组是一种最简单的多维数组．

多维数组类型定义的格式与二维数组类型定义格式相似．例如，定义三维数组类型的格式为

```
数组类型名=array[下标类型1,下标类型2,下标类型3]of元素类型
```

多维数组元素的引用方法也类似于二维数组．例如，引用三维数组元素的格式为

```
数组变量名[下标表达式1,下标表达式2,下标表达式3]
```

与一维数组、二维数组一样，对于多维数组一般也只是使用数组元素，很少使用数组变量．当两个多维数组变量类型相同时，可以进行整体赋值．

虽然也可以定义多维数组常量，但在实际编程中这种情况较少见，故在此不作介绍．

[例6.20]　某服装店经销“长江”和“黄河”2种品牌的服装，每种品牌的服装有“俊士”、“靓女”、“寿翁”和“福婆”4种款式，每种品牌、款式的服装又分全棉、全毛、混纺3种面料和大、中、小3种规格．请写一个简易库存管理程序，它的功能是：在每次进货业务和销售业务发生后，对各种服装的库存量进行修改，并且可以随时输出各种服装的库存量．

[解题思路]

用一个4维数组count来记录各种服装的库存量．数组中，第1个下标取值'1'和'2'（表示品牌），第2个下标取值从'1'到'4'（表示款式），第3个下标取值从'1'到'3'（表示面料），第4个下标取值也是从'1'到'3'（表示规格）．

每当发生进货业务和销售业务时，首先输入4个表示品牌、款式、面料和规格的数字到变量i1、i2、i3和i4中，然后再输入数量n．若是进货业务，则令count[i1,i2,i3,i4]加n；若是销售业务，则令count[i1,i2,i3,i4]减n．

程序中定义了4个一维数组常量s1、s2、s3、s4，它们的每个元素都是字符串（保留字string表示字符串类型，在6.4节中将作详细介绍）．

[程序]

```
program ch620;
 type  type1='1'..'2'; {用于说明品牌}
       type2='1'..'4'; {用于说明款式}
       type3='1'..'3'; {用于说明面料}
       type4=type3; {用于说明规格}
 const
  s1:array[type1]of string[4]=('长江','黄河');
  s2:array[type2]of string[4]=('俊士','靓女','寿翁','福婆');
  s3:array[type3]of string[4]=('全棉','全毛','混纺');
  s4:array[type4]of string[2]=('大','中','小');
 var count:array[type1,type2,type3,type4]of integer;
     i1,i2,i3,i4,ch:char;
     n:integer;
 begin
  {初始化}
  for i1:='1' to '2' do
   for i2:='1' to '4' do
    for i3:='1' to '3' do
     for i4:='1' to '3' do
      count[i1,i2,i3,i4]:=0;
  repeat
   {显示菜单}
   writeln('0…结束');
   writeln('1…进货');
   writeln('2…销售');
   writeln('3…查询');
   writeln('4…列表');
   write('请输入功能号(0..4)');
   readln(ch);
   if (ch>'0')and(ch<'4') then
   begin
    {显示品牌、款式、面料、规格的编号与中文名称对照表}
    writeln;
    writeln('品牌:1.',s1['1'],'  2.',s1['2']);
```

```
        writeln('款式:1.',s2['1'],'  2.',s2['2'],'  3.',s2['3'],'  4.',s2['4']);
        writeln('面料:1.',s3['1'],'  2.',s3['2'],'  3.',s3['3']);
        writeln('规格:1.',s4['1'],'  2.',s4['2'],'  3.',s4['3']);
        {输入某种服装的品牌、款式、面料、规格的编号}
        repeat
         write('请输入4个数字:');
         readln(i1,i2,i3,i4);
        until  (i1>'0')and(i1<'3')and
               (i2>'0')and(i2<'5')and
               (i3>'0')and(i3<'4')and
               (i4>'0')and(i4<'4'); {如果不正确,则重新输入}
        {显示这种服装的品牌、款式、面料、规格的中文名称}
        write('  品牌:', s1[i1],'  款式:', s2[i2],'  面料:', s3[i3],'  规格:',
s4[i4]);
        case ch of
         '1': {进货业务}
         begin
          write('  进货量:');
          readln(n);
          count[i1,i2,i3,i4]:=count[i1,i2,i3,i4]+n {增加库存}
         end;
         '2': {销售业务}
         begin
          write('  销售量:');
          readln(n);
          count[i1,i2,i3,i4]:=count[i1,i2,i3,i4]-n {减少库存}
         end;
         '3': {查库存量}
         begin
          write(' 库存量:',count[i1,i2,i3,i4]);
          writeln;
          write('请按回车键');
          readln {等待按回车键，以便让用户看清屏幕上显示的内容}
         end
```

```
    end
  end
  else if ch='4' {列表}
   then begin
    {列出现有服装库存量}
    writeln('                    库    存    表');
    writeln('    品牌    款式    面料    规格    库存量');
    for i1:='1' to '2' do
     for i2:='1' to '4' do
      for i3:='1' to '3' do
       for i4:='1' to '3' do
        if count[i1,i2,i3,i4]>0    {库存量不为零}
        then  begin
          write(s1[i1]:8,s2[i2]:8,s3[i3]:8,s4[i4]:7);
          writeln(count[i1,i2,i3,i4]:8)
        end;
    writeln;
    write('请按回车键');
    readln
   end
 until ch='0'
end.
```

6.4 字 符 串

6.4.1 字符串直接量和字符串常量

用一对单引号括起来的字符序列称为**字符串直接量**，简称字符串．序列中字符的个数称为字符串的长度，长度为零的字符串称为空串．在前面几章的例题中已多次出现字符串直接量(都是用于输出提示信息)．

字符串常量是一个在常量定义部分定义的标识符，是某个字符串直接量的名字．例如：

```
const
```

```
title='学号　姓名　成绩'
```

在这个定义中，等号的左边是一个字符串常量，等号的右边是一个字符串直接量。有了这个常量定义，在程序的执行部分就可以出现语句

```
writeln(title)
```

它的作用和语句

```
writeln('学号　姓名　成绩')
```

完全相同。

6.4.2　字符串类型和字符串变量

和其它类型的变量一样，字符串变量在使用之前也要进行说明。用于说明字符串变量的类型是字符串类型。字符串类型属于非标准类型，是需要预先进行定义的。

字符串类型定义的格式为

```
字符串类型名=string[长度]
```

其中，string是保留字；方括号中的长度是一个整数类型的常量或直接量，它规定这种类型的字符串中最多可以有几个字符。在Pascal语言中，任何字符串的长度值不得超过255。

如果长度值等于255，则类型定义可以简化为

```
字符串类型名=string
```

字符串变量的说明方法与其它变量的说明方法完全相同。

例如，假设课程名称不超过8个字符，则说明一个用于存放课程名称的字符串变量cns可按如下方式进行：

```
type
 coursename=string[8];
var
 cns:coursename;
```

或者

```
var cns:string[8];
```

字符串变量中每个字符都有一个相应的下标。上例中，变量cns中最多可以放入8个字符。我们可以通过cns[1]，cns[2]，…，cns[8]使用这些字符。在这一点上，字符串变量cns很像一个如下说明的、元素类型是字符型的一维数组cna：

```
var
  cna:array[1..8] of char;
```

但是，要注意：数组变量cna不可以用read/write语句输入/输出，而字符串变量cns却是可以的。

6.4.3 字符串的运算

对字符串（包括字符串直接量、字符串常量、字符串变量，以及值为字符串类型的表达式）的运算只有连接运算、关系运算和赋值运算3种。

1. 连接运算

字符串的连接运算符是“+”。这是个双目运算符，“+”号的两边都是字符串。运算的结果是把两个字符串拼接在一起，构成一个新的字符串。

例如：

连接表达式	结果
'Turbo'+'pascal'	'Turbopascal'
'1234'+'*'+'5678'	'1234*5678'

由连接运算符和字符串构成的表达式称为**字符串表达式**。

2. 关系运算

关系运算符<、<=、=、<>、>、>=可以用于字符串的比较，两个字符串的比较是按字典顺序进行的。字符串关系运算的结果是一个布尔值。

例如，下列关系运算结果为true：

```
'ABC'='ABC';
'ABC'<>'abc';
'Pascal'>'pascal'
'&41'>'&21'
```

连接运算符和关系运算符可以出现在同一个表达式中。在这种表达式中，连接运算符的优先级高于关系运算符，要先进行连接运算，后进行关系运算。因此，这种表达式属于关系表达式。

3. 赋值运算

可以把一个字符串表达式赋值给一个字符串变量。如果字符串变量被赋予过长的值，那么后面的字符将被截掉。

例如：

```
var cn: string[8];
…
cn:='math'; {cn的实际长度为4,其值等于'math'}
cn:='TurboPascal'; {cn的实际长度大于最大长度8,其值等于TurboPas'}
```

6.4.4　标准过程和标准函数

为了便于对字符串的处理，Turbo Pascal系统提供了8个标准过程和标准函数。

1. delete过程

格式：delete(s,i,k)

功能：删除从字符串s的第i个字符开始的k个字符。其中，s必须是字符串变量。如果i+k的值大于s的实际长度，则删除从第i个字符开始的后面所有字符.

例如，若cn='physics'，则delete(cn,5,3)和delete(cn,5,6)的结果都将使cn='phys'。

2. insert过程

格式：insert(s,t,i)

功能：把字符串s插入到字符串t中第i个字符开始的位置上。其中，t必须是字符串变量；s可以是字符串表达式；i是整型表达式。如果i的值大于t的实际长度，则将s连接在t的后面；如果插入后新字符串的实际长度大于t的最大长度，则右边多出的字符将被截掉.

例如，若cn='phycs'，则insert('si',cn,4)的结果是使t='physics'。

3. str过程

格式：str(v,s)

功能：把数值表达式v的值转换成字符串并存入字符串变量s中。其中，v是一个可带参数的整型或实型表达式。这个语句除了计算结果是存入变量s中而不是输出到屏幕上以外，效果与write(v)语句一样。

例如，若 v=12345，则str(v:6,s)使 s=' 12345'；若v=3.14E4，则str(v:6,s)使s=' 31400'。

4. val过程

格式：val(s,v,c)

功能：把字符串s转换成对应的数值，存入变量v中。其中，s是字符串；v是整型或实型变量；c是整型变量，用来检查转换结果是否正确，如果转换成功，则c的值为0，否则c指出第一个出错字符位置。

例如，若s='123'，且v是整型变量，则val(s,v,c)使v的值为123、c的值为0；若s='120+130'，则val(s,v,c)使v的值无定义、c的值为4；若s='3.14E4'，且v是个实型变量，则val(s,v,c)使v的值为31400、c的值为0。

5. copy 函数

格式：copy(s,i,k)

功能：返回字符串s中从第i个字开始的连续k个字符构成的子串。其中，s是字符串；i和k是整型表达式。如果i的值大于s的实际长度，则返回一个空串（即不含任何字符的字符串）。如果i+k的值大于s的实际长度，则返回的子串由s中从第i个字符开始的后面所有字符构成。

例如，若s='Pascal'，则copy(s,3,2)返回值为'sc'；copy(s,4,7)返回值为'cal'；copy(s,8,1)返回值为''（即空串）。

6. concat函数

格式：concat(s1, s2, …, sn)

功能：连接字符串s1，s2，…，sn。其作用与字符串表达式

s1+s2+…+sn

完全相同。

例如，concat('Turbo',' ','Pascal')的返回值为'Turbo Pascal'。

7. length函数

格式：length(s)

功能：返回字符串表达式s的长度值（一个整数）。

例如，length('Pascal')的返回值为6。

8. pos函数

格式：pos(s,t)

功能：返回字符串s在字符串t中第一次出现的位置号（一个整数）。

例如，若t='Goodbye'，则pos('bye',t)的返回值为5；pos('o',t)的返回值为

2；pos('good', t)的返回值为0。

6.4.5　应用举例

[例6.21]　输入一个由若干单词组成的文本行（长度不超过255个字符，每个单词之间用若干空格隔开），统计文本行中单词的平均长度。

[解题思路]

把文本行读到字符串变量s中，从前往后检查其中的每一个字符：若当前字符是字母，则将字母总数ic加1；若当前字符是字母，而前一个字符是空格，则将单词总数wc加1。最后，输出平均值ic/wc。

用nowc存放当前正在检查的字符，prec存放前面一个字符。在往nowc中存入下一个字符之前，先把nowc中的内容送入prec，使prec和nowc中的内容始终保持前后相邻关系。

[程序]

```
program ch621;
 const bland=' ';
 var s:string;
      nowc,prec:char;
      len,i,lc,wc:integer;
 begin
  writeln('请输入一个文本行');
  read(s);
  len:=length(s);
  lc:=0;wc:=0;
  prec:=bland;
  for i:=1 to len do
  begin
   nowc:=s[i]; { 当前要检查的字符存入nowc中 }
   if nowc<>bland then
   begin
    lc:=lc+1;    { 字母总数加1 }
    if prec=bland { 前面一个字符是空格 }
    then wc:=wc+1   { 字母总数加1 }
   end;
```

```
      prec:=nowc        { 使prec和nowc保持前后关系 }
    end;
    writeln;
    write('单词平均长度:',lc/wc:5:2)
  end.
```

[运行实例]

```
请输入一个文本行
This is a book
单词平均长度: 2.75
```

[例6.22]　输入3个字符串r、s、t，把字符串t中所有形如s的子串替换成r。

[解题思路]

用函数length(s)测出s的长度k；用函数pos(s,t)测出s在t中的位置i；把t[i]~t[i+k-1]替换成r；对t的右边剩余部分用同样的方法处理。

[程序]

```
program ch622;
 var s,t,r,x:string;
     i,k:integer;
 begin
  write('输入r:');readln(r);
  write('输入s:');readln(s);
  write('输入t:');readln(t);
  k:=length(s);
  x:='';    {空串赋值给x}
  repeat
   i:=pos(s,t);       { 检测s在t中首次出现的位置 }
   if i>0 {如果t中有s}
   then begin
    x:=x+copy(t,1,i-1)+r;   { 拼接 }
    t:=copy(t,i+k,255)      { 截取右边剩余部分 }
   end
  until i=0;
  x:=x+t;
  write('替换结果:',x)
end.
```

[运行实例]

输入r:XO

输入s:OXO

输入t:XOOXOXOXOX

替换结果:XOXOXXOX

[例6.23] 输入一个字符序列，判别其是否为整数、十进制实数或一般的字符串。

[解题思路]

把读入的字符串存入变量s中，从前往后检测每一个字符。

用枚举变量state表示程序在执行过程中所处的状态，它取A~F六个值，初始值为A．程序的执行过程是一个状态转换过程：

● 在A状态，若遇到空格字符，则程序滞留A状态不变；若遇正负号，则进入B状态；若遇数字，则进入C状态；若遇其它字符，则进入F状态。

● 在B状态，若遇空格字符，程序滞留B状态不变；若遇数字，则进入C状态；若遇其它字符，则进入F状态。

● 在C状态，若遇数字，则程序滞留C状态不变；若遇小数点，则进入D状态；若遇其它字符，则进入F状态。

● 在D状态，若遇数字，则进入E状态；若遇其它字符，则进入F状态。

● 在E状态，若遇数字，则程序滞留E状态不变；若遇其它字符，则进入F状态。

● 在F状态，终止程序执行。

以上所述是一个重复执行的过程。当循环结束时，根据state的值决定输出内容：如果state的值是C，则输出'整数'；如果state的值是E，则输出'实数'；如果state的值是A,B,D,F中的任何一个，则输出'其它'.

状态转换过程如图6.2所示。

[程序]

```
program ch623;
 var state:(A,B,C,D,E,F);
      s:string;
      len,i:integer;
 begin
  write('输入一个字符序列:');
  read(s);
  len:=length(s);
```

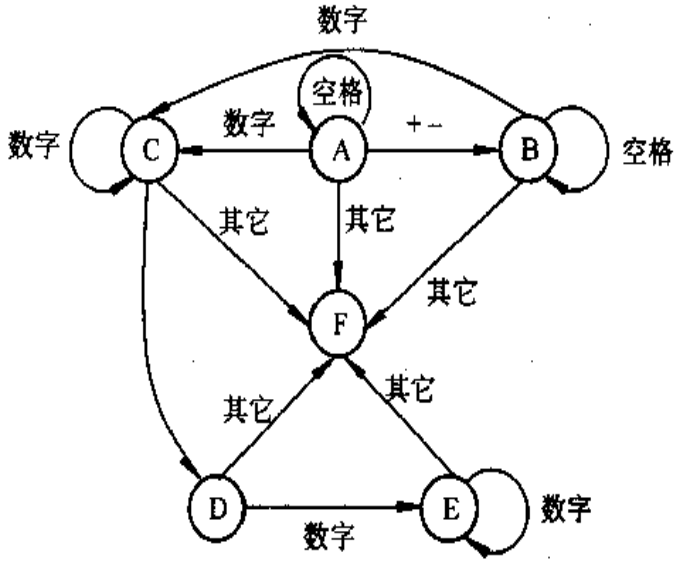


图6.2　状态转换过程

```
i:=1; {i指示当前正在检测的字符}
state:=A;
while i<=len do
begin
 case state of
  A:if (s[i]='+')or(s[i]='–')
     then state:=B
     else if (s[i]>='0')and(s[i]<='9')
          then state:=C
          else if s[i]<>' '
               then state:=F;
  B:if (s[i]>='0')and(s[i]<='9')
     then state:=C
     else if s[i]<>' 'then state:=F;
  C:if s[i]='.'
     then state:=D
     else if (s[i]<'0')or(s[i]>'9')
          then state:=F;
  D:if (s[i]>='0')and(s[i]<='9')
     then state:=E
     else state:=F;
  E:if (s[i]<'0')or(s[i]>'9')
```

```
          then state:=F;
        F:i:=len { 准备退出while循环 }
      end;
      i:=i+1      {准备检测下一个字符}
    end;
    write('判定结果:');
    case state of
      C:write('整数');
      E:write('实数');
      A,B,D,F:write('其它')
    end
  end.
```

[运行实例]

输入一个字符序列: 1998
判定结果:整数
输入一个字符序列: 1998.12
判定结果:实数
输入一个字符序列: 1998.12.30
判定结果:其它

[例6.24] 输入一个数字串s和整数k（k小于数字串s的长度），从s中删去k个数字，使剩余数字在保持相对位置不变的情况下构成一个值最小的整数。例如，s='19990608'，k=4，处理结果为：608。

[解题思路]

把读入的数字串存入变量s中，从前往后检测相邻字符s[i]和s[i+1]，如果有s[i]>s[i+1]，就将s[i]删除，k减1，然后回到串首重新开始检测；如果所有相邻字符都是s[i]<=s[i+1]，则将串尾k个字符都删除之，k减至0。

在删去了k个数字后，如果串首有'0'字符，则将这些无效数字都删除之。

[程序]

```
program ch624;
 var s:string;
      i,k,n:integer;
 begin
  write('输入一个数字串:');
  read(s);
```

```
  write('输入一个整数:');
  read(k);
  n:=length(s);{ 求原数字串长度 }
  while k>0 do { 删k个数字 }
  begin
   i:=1;
   while (i<n)and(s[i]<=s[i+1])
    do i:=i+1; { 找满足条件s[i]>s[i+1]的s[i] }
   if i=n { 没找到 }
   then { 删右起k个元素 }
   begin
    delete(s,n-k+1,k);
    k:=0;
    n:=n-k
   end
   else { 删左起第i个元素 }
   begin
    delete(s,i,1);
    k:=k-1;
    n:=n-1
   end
  end;
  { 删串首多余的零 }
  while (n>1)and(s[1]='0') do
  begin
   delete(s,1,1);
   n:=n-1
  end;
  write('处理结果:',s)
 end.
```

[运行实例]

```
输入一个数字串:12311998
输入一个整数:2
处理结果:111998
```

输入一个数字串:123001998
输入一个整数:3
处理结果:1998

习 题 六

6.1 填空题。

(1) 简单变量是指_____。

(2) 数组是由_____构成的。

(3) 在定义数组类型时,下标类型只能是_____。

(4) 对数组元素可以进行哪些运算取决于_____类型。

(5) 两个数组变量进行整体赋值的条件是_____。

(6) 二维数组的第一个下标是___下标,表示数组有多少_____。

(7) 用array[-10..10,-5..2,1..4]of integer 类型说明的数组变量有____个元素。

(8) 字符串的长度不得超过_____。

(9) 如果字符串变量被赋予过长的值，则_____。

(10) 两个字符串的比较是按_____顺序进行的。

6.2 下列类型定义都是错误的,请说明理由。

(1) type T1:array[1..max]of real

(2) type T2=array[100..1]of real

(3) type T3=array[1.5..10.5]of real

(4) type T4=array[integer]of real

(5) var T5=array[10..10]of real

(6) type T6=array[1..'9']of real

6.3 阅读下面这个不完整的程序，根据程序的功能，将其补充完整。

{功能:从输入的10个实数中,找到并输出与其平均值最接近但不大于平均值的那个实数}

```
var
  n,k:integer;
  a:array[1..10]of real;
  b,s:real;
begin
  for n:=1 to 10 do read(a[i]);
```

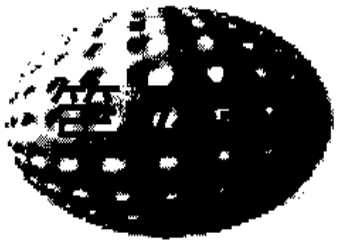
```
        [          ]
for k:=1 to 9 do s:=s+a[k];
s:=s/10;n:=1;
for k:=2 to 10 do
if [          ] then n:=k;
writeln( [          ] )
end.
```

6.4　写一个程序:输入全班n(n<50)位同学用拼音表示的姓名,检查班中是否存在姓名拼音相同的同学。若存在姓名拼音相同的同学,则列表输出这些拼音及其重复出现的次数。

6.5　写一个程序:根据从键盘输入的n的值，显示螺旋式数字方阵。例如,当n=4时,要求显示

```
 1  12  11  10
 2  13  16   9
 3  14  15   8
 4   5   6   7
```

6.6　单人纸牌游戏按如下规则进行:把拿掉了大王和小王的52张扑克牌分成4堆，每堆13张。翻开其中任意一堆最上面的一张牌，如果是红桃，就把它插入第一堆的最下面，接着翻开第一堆最上面的一张牌；如果是方块，就把它插入第二堆的最下面，接着翻开第二堆最上面的一张牌；如果是黑桃，就把它插入第三堆的最下面，接着翻开第三堆最上面的一张牌；如果是梅花，就把它插入第四堆的最下面，接着翻开第四堆最上面的一张牌；如果在某一堆的下面插入一张牌后，发现这一堆13张牌已全部翻开，则游戏结束。用已全部翻开牌的堆数作为一次游戏的得分。试编写模拟这个游戏的程序。



子程序

一个复杂程序一般都要完成若干个主要的任务，每个主要的任务又可分成若干个子任务。在动手写程序前，对程序要实现的功能进行逐层分解，先设计出实现各子任务的程序段，然后将它们组合起来，构成一个完整的程序。这种程序设计方法称为**结构化程序设计**。用这种方法设计的程序称为**结构化程序**。

在结构化程序中，对于如上所述的用于完成某种任务的程序段，通常用一个合适的标识符予以命名。经过命名的程序段称为**子程序**。

一个结构化程序一般都是由一个**主程序**和若干个子程序构成的。主程序通过调用子程序来实现相应的功能，各子程序之间也可以相互调用。但是，任何一个子程序都不可以去调用主程序，因为程序的执行总是从执行主程序开始的。前面几章中所编写的程序都是只有主程序而没有子程序。

采用结构化程序设计方法设计的程序，不仅层次清晰、便于阅读、便于修改，而且可以避免对同一功能程序段的重复编写。例如，要根据m和n的值计算d：

$$d = \frac{m!}{n!(m-n)!}$$

可以先设计一个计算k！的通用子程序，然后分别用k等于m，k等于n和k等于（m-n）去调用这个子程序，最后对3次调用结果进行乘除运算即得d值。

在用Pascal语言编写的程序中，子程序有函数和过程两种形式。下面就来介绍这两种子程序。

7.1 函 数

函数是一个经过命名的程序段，它产生一个值。第1章已介绍过一些Pascal函数，这些函数可以在程序中直接引用，称为标准函数。如果想在程序中使用“自己的函数”，例如，计算k！的函数，则必须在程序的说明部分对这个函数进行定义：如何计算？函数名是什么？函数值是什么类型？等等。由程序设计者自己定义的函数称为自定义函数。在程序中，任何自定义函数在引用它之前都必须先给出其定义。

7.1.1 函数定义

函数定义由函数首部和函数体组成，其格式为

```
function 函数名(形式参数表):函数值类型;   { 函数首部 }
 …          { 说明部分 }
 begin                                  { 函数体 }
 语句组     { 执行部分 }
end
```

1. 函数首部

函数首部以保留字function开头，function和后面的函数名之间至少要有一个空格。

函数名是一个由程序设计者自己选定的标识符，是用来存放函数值的，其类型由冒号后面的函数值类型指明。数组、集合、记录和文件不可以用作函数值类型。

形式参数表中列出的是若干个用分号分隔的形式参数说明。对于函数来说，一般情况下每个形式参数说明具有如下格式：

```
形式参数名1,形式参数名2, …, 形式参数名k:类型标识符
```

形式参数类似于一般数学函数中的“自变量”，它们为函数子程序加工处理数据提供初始值。

2. 函数体

函数体由说明部分和执行部分组成.

说明部分对仅在函数中使用的量加以说明，在这里，可以出现常量定义、类型定义和变量说明，也可以出现函数定义或过程定义．在函数体内说明的各种量，其作用范围只局限于函数体内部，即只能在这个函数体中使用，它与函数体外可能出现的同名量无关.

执行部分是一个由begin和end括起来的复合语句，它描述了函数值的计算过程．在这个复合语句中至少要有一个给函数名赋值的语句.

函数体和程序体非常相似，只有两点不同：

① 函数体以分号结束，程序体以句号结束；

② 函数体中至少要有一个给函数名赋值的语句，程序体中不可以有给程序名赋值的语句.

7.1.2 函数调用

可以在任何与函数值类型兼容的表达式中调用函数，也就是说函数的调用只能出现在允许表达式出现的地方．调用方式与调用标准函数的方式相同，其格式为

函数名(实在参数表)

实在参数表中各实在参数（简称实参）之间用逗号隔开，它们和函数定义中的形式参数（简称形参）按序一一对应，个数相等，类型兼容．实在参数的名字与对应形式参数的名字无关，既可以用相同的名字，也可以用不相同的名字，在程序中它们总是代表不同的存储单元.

7.1.3 实例

[例7.1] 编写一个求 k ! 的函数，调用此函数计算：

$$d = \frac{m!}{n!(m-n)!}$$

其中，n<m<8.

[程序]

{ 为了便于叙述，下面这个程序每一行的前面都加了行号．实际输入这个程序时，这些行号应该去掉 }

```
1   program ch71;
2    var m,n:integer; {全程变量说明}
3        a,b,c,d:integer;
4   function fac(k:integer):integer;
5    var i:integer;{局部变量说明}
6        t:integer;
7    begin  {函数执行部分}
8     t:=1;
9     for i:=2 to k do t:=t*i;
10    fac:=t
11   end;
12  begin {主程序执行部分}
13   writeln('输入m和n:');
14   read(m,n);
15   if (n>=0) and (n<m) and (m<8)
16   then begin
17          a:=fac(m);
18          b:=fac(n);
19          c:=fac(m-n);
20          d:=a div (b*c);
21          write('计算结果:');
22          writeln('d=',d)
23         end
24   else writeln('输入数据不正确')
25  end.
```

[运行实例]

```
输入m和n:5  3
计算结果:d=10
```

在这个程序中，第2行到第11行是主程序的说明部分，第12行到25行是主程序的执行部分。在主程序的说明部分，第2行和第3行说明了程序将要用到的6个变量m、n、a、b、c、d，这些变量称为**全程变量**。所谓全程变量是指在主程序的执行部分和子程序的执行部分都可以使用的变量。从第4行到第11行是函数fac的定义，用于计算 k!，它有一个整数类型的形式参数k。第5行和第6行是函数的说明部分，在此说明了为完成k! 计算所需要的变量i

和t，这两个变量称为局部变量。所谓局部变量是指只能在相应的子程序的执行部分中使用的变量。

对这个程序的编译是从第1行开始的。系统逐行检查程序是否存在语法错误，在不存在语法错误的前提下，系统要为全程变量m、n、a、b、c、d和函数名fac分配存储单元，但不为形式参数k和局部变量i、t分配存储单元。

函数执行部分虽然写在主程序执行部分之前，但执行这个程序却是从第12行开始的。当执行到第14行时，通过键盘读两个整数(例如，5和3)存入与全程变量m和n对应的存储单元中。当执行到第17行时，遇到第一次函数调用，系统将中断主程序的执行，转去执行子程序。在此期间，系统主要做4件事：

① 为局部变量i、t和形式参数k分配临时存储单元；

② 将实在参数m的值存入与形式参数k所对应的临时存储单元中；

③ 执行第7行到第11行的复合语句；

④ 将计算结果120（5!=120）返回到调用处（第17行赋值号的右边），并释放所有的临时存储单元。

至此，第一次函数调用结束，系统从主程序的中断处开始，继续往下执行主程序。首先是将函数值120存入全程变量a所对应的存储单元中，然后从第18行开始往下执行。第二次、第三次函数调用情况与第一次相似，都有一个分配临时存储单元、把实在参数的值赋值给对应的形式参数、执行复合语句、返回计算结果并释放临时存储单元的过程。与第一次调用不同的是，第二次调用时实在参数值为3，函数计算结果为6；第三次调用时实在参数值为2，函数计算结果为2。

第一次调用函数fac前后，存储单元的变化情况如图7.1所示。

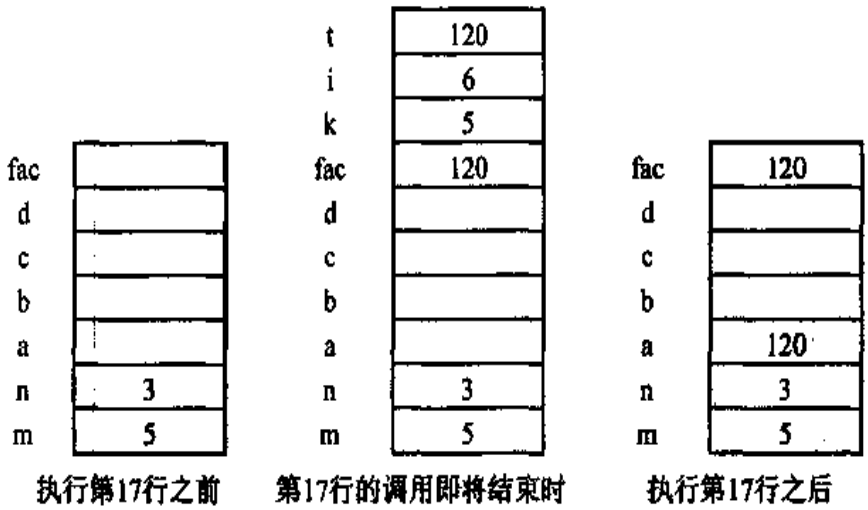


图7.1　调用fac(5)过程中存储单元的变化情况

7.2 过 程

函数子程序的运行结果是一个简单类型的值．有时，希望子程序能产生多个值（例如，从一组数中找出最大值和最小值）或完成某些与计算无关的操作（例如，对一维数组中的元素重新排序），这时通常使用过程子程序。

和函数一样，要在程序中使用“自己的过程”，也必须在程序的说明部分给出其定义．

7.2.1 过程定义

过程定义由过程首部和过程体组成，其格式为

```
procedure 过程名(形式参数表);      { 过程首部 }
 …            { 说明部分 }
begin                                  { 过程体 }
 语句组       { 执行部分 }
end
```

1. 过程首部

过程首部以保留字procedure开头，procedure和后面的过程名之间至少要有一个空格．

过程名是一个由程序设计者自己选定的标识符，用于标识这个子程序，它没有“值”的概念(而函数名有一个值)．

形式参数表中列出的是若干个用分号分隔的形式参数说明．对于过程来说，形式参数说明一般有两种格式．

格式1：

```
形式参数名1, 形式参数名2, …, 形式参数名k:类型标识符
```

格式2：

```
var 形式参数名1,形式参数名2, …, 形式参数名k:类型标识符
```

用第一种格式说明的形式参数称为数值形参；用第二种格式说明的形式参数称为变量形参．

数值形参用于为子程序加工处理数据提供初始值．变量形参虽然也可以用于为子程序提供所需处理数据的初始值，但主要用于将子程序的处理结果传递给调用者．在7.3节中将详细讨论这两种形式参数．

过程子程序可以没有形式参数，这种不带参数的过程称为无参过程．无参过程的过程首部格式为

```
procedure 过程名
```

注：严格地说，函数子程序也可以没有形式参数，但是，这种没有“自变量”的无参函数在实际应用中不常见。

2. 过程体

过程体和函数体非常相似，唯一的区别是：过程体中没有也不可以有给过程名赋值的语句．

7.2.2　过程调用

过程调用是以语句的形式出现在程序中的，这种语句称为过程调用语句．带参数过程的过程调用语句，其格式为

```
过程名(实在参数表)
```

实在参数表中各实在参数之间用逗号分隔，它们和过程定义中的形式参数按序一一对应，个数相等，类型兼容．如果形式参数是数值形参，则与之对应的实在参数可以是表达式；如果形式参数是变量形参，则与之对应的实在参数就必须是变量，而不能是一般的表达式．

无参过程的过程调用语句只有一个过程名，其格式为

```
过程名
```

7.2.3　实例

[例7.2]　将以秒为单位的时间转换成小时、分、秒的形式，用过程实现．

[程序]

{为了便于叙述，下面这个程序每一行的前面都加了行号．实际输入这个程序时，这些行号应该去掉}

```
1  program ch72;
```

```
2   var   t,h,m,s: integer;
3   procedure   time(t:integer;var h,m,s:integer);
4    begin {过程执行部分}
5     h:=t div 3600;
6     m:=(t mod 3600) div 60;
7     s:=(t mod 3600)   mod 60
8    end;
9   begin   {主程序执行部分}
10   write ('输人以秒为单位时间:');
11   read(t);
12   if t>0 then
13    begin
14     time(t,h,m,s);
15     writeln;
16     writeln('转换成:',h,'小时',m '分',s,'秒')
17    end
18  end.
```

[运行实例]

输人以秒为单位时间: 5432

转换成:1小时30分32秒

在这个程序中，第2 行至第8行是主程序的说明部分，第9行至第18行是主程序的执行部分．在主程序的说明部分，第2行说明了4个全程变量t、h、m和s；第3行至第8行说明了一个带形式参数的过程子程序time，这个过程子程序没有自己的说明部分．在过程首部中出现的4个形式参数中，t是数值形参，h、m和s是变量形参．过程的作用是将t秒转换成h小时m分s秒．由于输出语句是在主程序中，为了将过程的处理结果传递给调用者（主程序），h、m和s这3个形式参数用的是变量形参．

程序的编译从第1行开始，逐行进行。编译成功后，系统给全程变量t、h、m和s分配存储单元．

程序的执行从第9行开始．执行到第11行语句时，从键盘读一个整数（例如5432）存人全程变量t所对应的存储单元．执行到第14行的过程调用语句，系统将中断主程序的执行，转去执行子程序．在此期间，系统主要做4件事：

① 为形式参数t,h,m和s分配临时的存储单元；

② 将实在参数t的值5432存人与数值形参t所对应的临时存储单元，将

实在参数h、m和s的地址存入与变量形参h、m和s所对应的临时存储单元中；

③ 执行第4行至第8行的复合语句；

④ 释放所有的临时存储单元，返回调用处。

至此，过程调用结束．程序从中断处开始，继续往下执行．

过程的执行部分是3个给变量形参赋值的语句，表达式的值不是直接存入与形式参数h、m和s对应的临时存储单元，而是利用系统的间接寻址功能，将表达式的值存到与实在参数h、m和s对应的存储单元中。

程序执行期间存储单元的变化情况如图7.2所示．

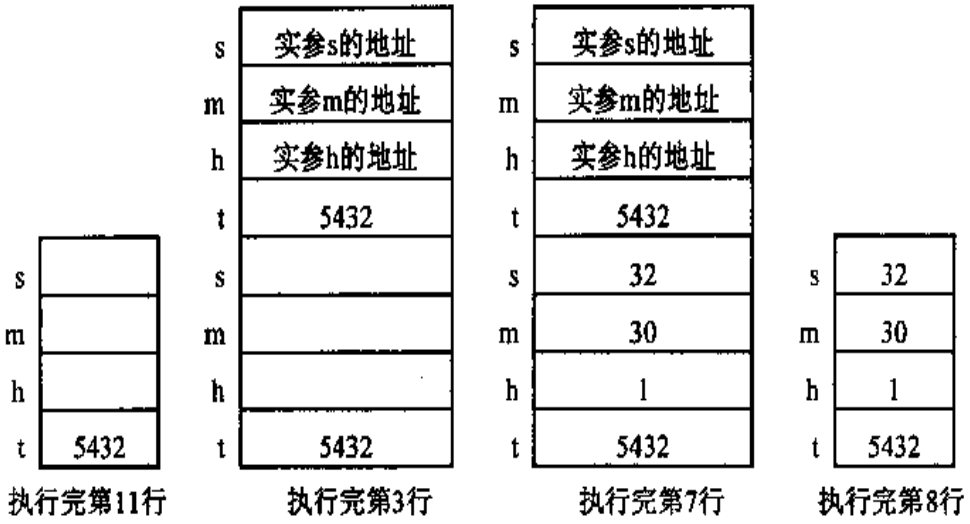


图7.2　程序执行期间存储单元的变化情况

7.3　变量和参数

前面两节中提到的全程变量和局部变量、数值形参和变量形参等概念是非常重要的．这一节通过几个简单例子来进一步研究这些概念之间的联系与区别．

[例7.3]　写一个程序，通过子程序调用实现将整型变量x和y的值相互交换。

1. 用不带参数的过程实现

[程序]

{ 为了便于叙述，下面这个程序每一行的前面都加了行号．实际输入这个程序时，这些行号应该去掉 }

```
1  program ch73a;
```

```
 2   var x,y:integer;
 3   procedure swap;
 4    var t:integer;
 5    begin
 6     t:=x;x:=y;y:=t
 7    end;
 8   begin
 9    write('输入:');
10    read(x,y);
11    swap;
12    write('输出:');
13    writeln(x,'  ',y)
14   end.
```

[运行实例]

输入:1 2

输出:2 1

在过程swap中说明的变量 t 是个局部变量，它是为实现交换x、y的值而引入的。把这个变量放到第2行，即作为全程变量来说明，虽然也可以，但不合理。因为主程序的执行部分不需要用到这个变量。一般地说，为了实现子程序的功能所引入的那些中间变量、循环控制变量等，都应作为局部变量，放在子程序中说明。

如果某人画蛇添足，把第4行的局部变量说明改为

```
var  t,x,y:integer;
```

将会出现什么结果呢？此时，局部变量x、y和全程变量x、y同名，在语法上没有错误，但是这个swap过程对主程序来说毫无意义。主程序调用这个过程时，它交换的是两个未经赋值的局部变量x、y的值，而不是由read语句输入的全程变量x、y的值。因此，程序输出结果和输入完全相同。

注意：

① 在主程序的执行部分，只能使用全程变量；在子程序的执行部分，不仅可以使用局部变量，而且可以使用全程变量。

② 当局部变量和全程变量同名时，在子程序的执行部分使用的是与全程变量同名的局部变量。

2. 用带参数的过程实现

[程序]

{为了便于叙述，下面这个程序每一行的前面都加了行号．实际输入这个程序时，这些行号应该去掉}

```
1  program ch73b;
2   var x,y:integer;
3   procedure swap(var x,y:integer);
4    var t:integer;
5    begin
6     t:=x;x:=y;y:=t
7    end;
8   begin
9    write('输入:');
10   read(x,y);
11   swap(x,y);
12   write('输出:');
13   writeln(x,'  ',y)
14  end.
```

[运行实例]

输入:1　2

输出:2　1

这个swap过程所带的两个形式参数x、y是变量形参．当执行到过程调用语句时，系统将实在参数x、y的地址传递给对应的形式参数x、y．在过程执行期间，对形式参数x、y的引用和赋值，实际上是对相应的实在参数的引用和赋值．因此，swap的作用是将与形式参数x、y对应的实在参数x、y的值互相交换．

形式参数和对应的实在参数不一定要用相同的名字．本例中，形式参数x、y和实在参数x、y同名只是一种巧合．如果swap过程中的两个参数用其它标识符，或者第一个形参用y，第二个形参用x，都不会影响这个程序的输出结果．

swap过程的形式参数x、y，如果不是用变量形参，而是用数值形参，即把第3行改为：

```
procedure swap(x,y:integer);
```

将会出现什么结果呢？当执行到过程调用语句时，系统将实在参数x、y的

值传递给对应的形式参数x、y。在进入过程体以后，实在参数和形式参数之间就不再有任何关系。过程体中交换的是形式参数x、y的值，其结果不影响对应的实在参数x、y。因此，输出结果和输入的完全相同。

在主程序和子程序之间（或子程序和子程序之间）传递数据有两种途径，一种是用全程变量（如本例的第一个程序）；另一种是用参数（如本例的第二个程序）。一般认为使用参数较好。因为这样可以提高子程序的独立性，便于阅读、修改程序。

在编写带参数的子程序时，如何确定某个形式参数是用数值形参还是用变量形参？一般地说，可以遵循以下原则：

● 如果这个参数只用于“输入”子程序所要处理的数据，则用数值形参；

● 如果这个参数不仅要用于“输入”，而且还要用于“输出”，或者只用于“输出”，即把子程序处理结果传递给调用者，则用变量形参。

注意：

① 与数值形参对应的实在参数可以是一个变量，也可以是一般的表达式；子程序对数值形参的任何处理都不会影响到对应的实在参数，即实在参数的值不会被子程序所改变。

② 与变量形参对应的实在参数只能是一个变量，不可以是一般的表达式；子程序对变量形参的处理实际上是在对应的实在参数上进行的，即实在参数的值可能会被子程序所改变。

[例7.4] 在一个字符序列中，我们把相同字符所构成的子序列称为平台．请写一个程序，其功能是，从键盘读入一个以句号结尾、长度任意的字符序列，分别统计由字符'a'所构成的平台的最大长度值，和由字符'b'所构成的平台的最大长度值．

[解题思路]

定义一个过程子程序count(t,n)，它的功能是统计当前遇到的由字符t所构成的平台的长度值i，若i大于迄今为止这种平台的最大长度值n，则更新n值．显然，n应该是变量形参。

[程序]

```
program ch74;
 var s:char;
     na,nb:integer;
 procedure count(t:char;var n:integer);
  { 读入一个t平台,更新这种平台的最大长度值n }
```

```
  var i:integer;
  begin
   i:=0;
   repeat
    i:=i+1;read(s)
   until s<>t; {直到读入一个与t不同的字符为止}
   if i>n then n:=i
  end;
 begin
  writeln('输入一个以句号结尾的字符序列');
  read(s);
  na:=0;nb:=0;
  while s<>'.' do
  begin
   case s of
    'a':count('a',na);
    'b':count('b',nb)
   end;
   read(s)
  end;
  writeln('平台a的最大长度值为:',na);
  writeln('平台b的最大长度值为:',nb)
 end.
```

[运行实例]

输入一个以句号结尾的字符序列

abcdanbbcdajkbccaaabcaaaacfg.

a平台的最大长度值为:4

b平台的最大长度值为:2

然而，前面关于根据参数的作用决定选择数值形参或变量形参的一般原则，也有例外的情形．如果形式参数是一个数组，则不论其作用如何，一般都选用变量形参，因为这样可以提高程序的执行效率．

[例7.5]　写一个过程子程序，用于判别某同学的名字是否在n（n<100）个学生名册中．

设有常量及类型定义：

```
const maxn=100;
type index=1‥maxn;
     class=array[index] of string;
```

则可以设计如下过程子程序：

```
procedure find(s:class;n:index; {s是学生名册,n是实际人数}
               x:string;        {x是要查找的学生姓名}
               var t:boolean);  {若x在s中则返回true,否则返回false}
 var i:integer;
 begin
  i:=0; t:=false;
  repeat
   i:=i+1;
   if s[i]=x then t:=true
  until (i>n) or t
 end;
```

当调用这个过程时，系统要为它分配104个临时单元，如图7.3所示。

如果s用变量形参，即将过程首部改为

```
procedure find (var s:class;n:index; {s是学生名册,n是实际人数}
               x:string;        {x是要查找的学生姓名}
               var t:boolean);  {若x在s中则返回true,否则返回false}
```

则当调用这个过程时，系统只为它分配5个临时单元，如图7.4所示。s虽然是个数组，但只为它分配一个单元，用于存放对应实在参数（是个同类型数组）的首地址。在过程中访问形参数组元素s[i]实际上是访问实参数组的对应元素。

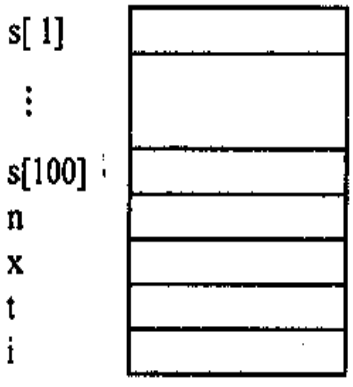


图7.3 数组作为数值形参

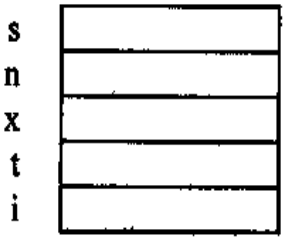


图7.4 数组作为变量形参

在进行子程序调用时，如果数组参数是个数值参数，则要把每个实参

数组元素传递给对应的形参数组元素．数组元素越多，这个传递过程所需时间也就越长。如果数组参数是个变量参数，则不管数组有多少个元素，都是只传递数组的首地址，因此，可以提高程序的执行效率．

[例7.6] 精确计算 1!+2!+3!+…+n!

其中，n≤50．

[解题思路]

引入两个表示大数的一维数组a、b（参考例6.9的用法），它们的左起第一个非零元素的位置分别用变量ka、kb表示．

定义过程子程序 mult(b,kb,m) 用于实现求数组b和整数m的乘积．

定义过程子程序 add(a,b,ka,kb) 用于实现将数组b累加到a数组中．

在主程序中，通过循环

```
for i:=1 to n do
 begin
  mult (b,kb,i);
  add(a,b,ka,kb)
 end;
```

完成计算任务．

[程序]

```
program ch76;
 const max=100;
       n=20;
 type numarr=array[1..max]of 0..9;
 var a,b:numarr;
     ka,kb,i:integer;

 procedure mult(var b:numarr; var kb:integer;
                m:integer);
{ 求b与m之积.kb是数组b左起第一个非零元素的下标 }
 var i,x:integer;
 begin
  x:=0;
  { 逐位相乘 }
  for i:=max downto kb do
  begin
   x:=b[i]*m+x;      { 求乘积与来自低位的进位之和 }
```

```
    b[i]:=x mod 10;    { 取个位数字 }
    x:=x div 10        { 产生新的进位 }
  end;
  { 处理进位 }
  while x>0 do
  begin
    kb:=kb-1;
    b[kb]:=x mod 10;
    x:=x div 10
  end
end;

procedure add (var a,b:numarr;
               var ka,kb:integer);
{ 把数组b累加到数组a中 }
 var i,k,x:integer;
 begin
  { 准备 }
  if ka<kb
  then k:=ka else k:=kb;
  x:=0; { 来自低位的进位 }
  { 逐位相加 }
  for i:=max downto k do
  begin
    x:=a[i]+b[i]+x;    { 求和 }
    a[i]:=x mod 10;    { 取个位数字 }
    x:=x div 10        { 产生新的进位 }
  end;
  { 处理进位 }
  while x>0 do
  begin
    k:=k-1;
    a[k]:=x mod 10;
    x:=x div 10
  end;
```

```
    { 令ka指示a数组左起第一个非零元素 }
    ka:=k
  end;

begin
  { 初始化 }
  for i:=1 to max do
  begin
    a[i]:=0;b[i]:=0
  end;
  b[max]:=1;
  ka:=max+1;kb:=max;
  { 求和 }
  for i:=1 to n do
  begin
    mult(b,kb,i);
    add(a,b,ka,kb)
  end;
  { 输出 }
  writeln;
  for i:=ka to max do write(a[i]:1)
end.
```

[运行结果]

2561327494111820313

7.4 并列与嵌套

这一节讨论子程序与子程序之间的相互关系.

在子程序和子程序之间存在两种基本关系，即并列与嵌套.

7.4.1 并列的子程序

在主程序(或某个子程序)的说明部分可以定义多个子程序，这些子程序称为并列子程序.

对于具有并列关系的子程序来说，后定义的可以调用先定义的．但是，先定义的不可以调用后定义的，除非作了超前引用说明．另外，每个子程序都可以自己调用自己．这种情况称为直接递归．

有关超前引用说明和直接递归的问题将在下一节介绍．

[例7.7] 在下面这个不完整的程序中，有两个并列的子程序SPA和SPB．由于SPB是在SPA之后定义的，所以SPB可以调用SPA，而SPA不可以调用SPB．

[程序]

```
program ch77;
 const a=10;
 var b:integer;
 procedure SPA(x,y:real);
  var i:char;
  begin{ SPA的执行部分 }
   { 这里，可以使用全程常量a和全程变量b; }
   { 可以使用参数x,y和局部变量i;还可以调用 }
   { SPA(即递归调用) }
  end;
 function SPB(z:integer):boolean;
  var j:real;
  begin{ SPB的执行部分 }
   { 这里，可以使用全程常量a和全程变量b; }
   { 可以使用参数z和局部变量j;可以调用SPA; }
   { 还可以调用SPB(即递归调用) }
  end;
 begin{ 主程序的执行部分 }
  { 这里，可以使用全程常量a和全程变量b; }
  { 可以调用过程SPA和函数SPB }
 end.
```

7.4.2 嵌套的子程序

在子程序的说明部分又出现子程序定义，这种情况称为嵌套．

[例7.8] 在下面这个不完整的程序中，子程序SPB是在子程序SPA的说

明部分中定义的；而子程序SPA又是在子程序SP的说明部分中定义的．

[程序]

```
program ch78;
 var a:integer;
 procedure SP(x:real);
  var b:char;
  function SPA(y:integer):boolean;
   var c:real;
   procedure SPB(z:integer);
    var d:real;
    begin{ SPB的执行部分 }
     {这里，可以使用变量a、b、c、d;可以使}
     {用参数x、y、z;可以调用SP、SPA和SPB}
    end;
   begin{ SPA的执行部分 }
     {这里，可以使用变量a、b、c;可以使}
     {用参数x、y;可以调用SP、SPA和SPB}
   end;
  begin{ SP的执行部分 }
     {这里，可以使用变量a、b;可以使}
     {用参数x;可以调用SP、SPA}
  end;
 begin{主程序的执行部分}
       {这里，可以使用变量a; }
       {可以调用SP}
 end.
```

在这个例子中，SPB称为内层子程序，SP称为外层子程序．SPA相对于SP是内层子程序，而相对于SPB则是外层子程序．相对于SP来说，SPA和SPB都是内层子程序．为了便于区分，我们把SPA称为SP的直接内层子程序，把SPB称为SP的非直接内层子程序．

主程序只能调用最外层的子程序SP，不可以调用内层子程序SPA和SPB．

外层子程序SP 只能调用它的直接内层子程序SPA，不可以调用它的非直接内层子程序SPB．

内层子程序可以调用嵌套自己的任何一个外层子程序，即SPB可以调用SPA和SP，SPA可以调用SP．这种情况称为间接递归，将在下一节介绍。

7.4.3 应用举例

[例7.9] 输入一个十进制数，将它转换成K进制数字串(K=2, 3, 4,…, 9)，并列表输出。

[解题思路]

把输入的十进制数分为整数x1和小数x2两部分。

把十进制整数x1转换成K进制整数，可通过反复进行“除K取余”的方法实现。例如，要将十进制整数129转换成八进制整数，可通过反复进行“除8取余”得到：

算式	余数	商
129÷8	1	16
16÷8	0	2
2÷8	2	0

经过3次“除8取余”操作后，商变为0，得到八进制整数201。

把十进制小数x2转换成K进制小数，可通过反复进行“乘K取整”的方法实现（见例5.14）。

[程序]

```
program ch79;
 var
  x,x2:real;
  x1,k:integer;
  s1,s2:string;
  i1,i2,i:integer;
 procedure getinteger(x1,k:integer);
 { 把正整数x1转换成k进制数字串并输出 }
  var i,j:integer;
      s:string;
  begin
   i:=0;
   while x1>0 do
    begin
```

```
      i:=i+1;
      s[i]:=chr(ord('0')+(x1 mod k)); {把余数转换成字符,存入s[i]中}
      x1:=x1 div k                    {取得商}
    end;
  if i=0 {待转换的整数x1是0}
  then write('0')
  else {按与得到这些余数的顺序相反的顺序输出这些余数}
   for j:=i downto 1 do write(s[j])
end;

procedure getdecimal(x2:real;k:integer);
{ 把正小数x2转换成k进制数字串并输出 }
 const maxlen=8;
 var d,i,j:integer;
      p:real;
      s:string;
 begin
  i:=0;
  while (x2>0) and (i<maxlen) do
   begin
    i:=i+1;
    p:=x2*k;
    d:=trunc(p);
    x2:=p-d;
    s[i]:=chr(d+ord('0'));
   end;
  write('.');
  for j:=1 to i do write(s[j])
end;

begin
 write('10进制数: ');
 read(x);
 for k:=9 downto 2 do
  begin
```

```
        write(k:2,'进制数: ');
        if x<0 then write('-');
        x1:=trunc(abs(x));
        getinteger(x1,k);
        x2:=abs(x)-x1;
        if x2>0 then getdecimal(abs(x2),k);
        writeln
      end
    end.
```

[运行实例]

```
10进制数:-10.5
 9进制数:-11.44444444
 8进制数:-12.4
 7进制数:-13.33333333
 6进制数:-14.3
 5进制数:-20.22222222
 4进制数:-22.2
 3进制数:-101.11111111
 2进制数:-1010.1
```

在这个程序中使用了一个新的标准函数chr，它的一般形式是chr(x)，它的功能是返回以整数x为序号的字符。

chr(x)和ord(x)是一对反函数，例如，

```
chr(48)='0'
ord('0')=48
chr(9+ord('0'))='9'
```

7.5 递 归

如前所述，内层子程序可以调用嵌套自己的任何一个外层子程序；任何一个子程序都可以自己调用自己。子程序的这种调用方式称为递归调用。

在现实生活中，有许多问题是以递归方式说明的，求解这类问题的Pascal程序用递归子程序最方便。

7.5.1　直接递归

所谓直接递归是指一个子程序自己调用自己的情况.

[例7.10]　写一个计算n!的递归函数.

n!可以用递归方式表示:

$$n!=\begin{cases}1 & n=1\\ n*(n-1)! & n>1\end{cases}$$

由于允许子程序自己调用自己、所以对于较小的n（n<8），可以设计出形式非常简单的计算n!的程序:

[程序]

```
program ch710;
 var f,n integer;
 function fac(n:integer):integer;
  begin
   if n=1
   then fac:=1
   else fac:=n*fac(n-1);
  end;
 begin
  read(n);
  if (n<1)or(n>7)
  then write('n值太小或太大')
  else begin
   f:=fac(n);
   write('n!=',f)
  end
 end.
```

在这个程序中，函数fac是以递归方式定义的，它的执行次数由在主程序中初次调用时的实在参数n值决定：若n=1，则只调用1次；若n=2，则递归调用2次；若n=k，则需递归调用k次.

子程序的递归调用次数称为递归深度，一般由初次调用时实在参数的值决定.子程序递归调用由外向里逐层进行，每深入一层，系统就要为它的形式参数和局部变量分配一组临时存储单元.在不同层次上的临时存储

单元尽管名字相同，但互不相干（因为它们代表不同的存储单元）．递归调用的返回过程也是逐层进行的，由里向外逐层返回，每返回一层就释放一组处于最里层的临时存储单元．

语句 f:=fac(4) 的执行过程如图7.5所示．图中每个方框代表一次函数调用，方框的左边是本次调用所建立的临时存储单元的值，方框的右边是本次调用结束时的返回值．

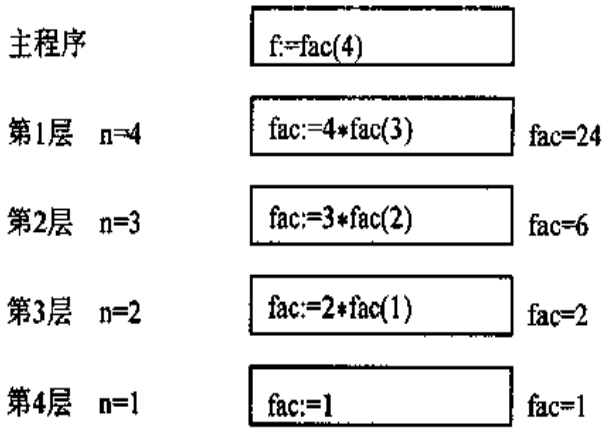


图7.5 赋值语句f:=fac(4)的执行过程

一个递归子程序必须有确切的含义，随着递归调用的深入，问题会越来越简单，最后可以结束．用以结束递归调用的条件称为边界条件．上面的例子中，当n=1时定义fac=1，就是边界条件．

[例7.11] 汉诺塔问题．如图7.6所示，有A、B、C三根细柱，A柱上有n个从小到大的圆盘(图中，n=3)，小的在上，大的在下．要求把这n个圆盘移到C柱上，在移动的过程中，可以借助B柱，每次只许移动一个圆盘，在任何一根细柱上都不允许出现大盘压住小盘的情况．请编写一个显示移动步骤的程序．

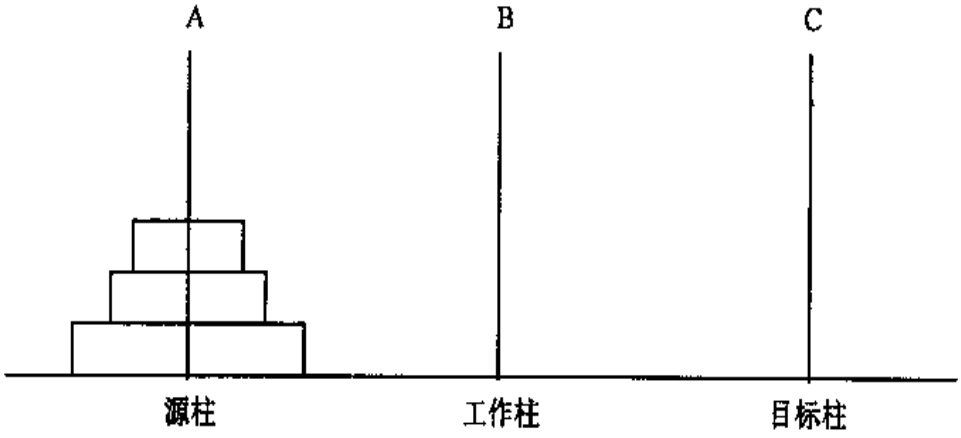


图7.6 汉诺塔问题

[解题思路]

原问题可以分解成3个子问题：

① 将A柱最上面的n－1个圆盘借助C柱移到B柱；

② 将A柱剩下的一个圆盘移到C柱；

③ 将B柱的n－1个圆盘借助A柱移到C柱。

显然，子问题①、③与原问题具有相同的性质，只是盘子数目少了一个，并且代表源柱、工作柱、目标柱的细柱代号不同而已。因此，可以用递归的方法来解决，结束递归的边界条件是n=0（当n=0时不做任何事情）。

[程序]

```
program ch711;
 var n:integer;
 procedure hanoi(n:integer;a,b,c:char);
  begin
   if n>0 then
    begin
     hanoi (n-1,a,c,b);
     writeln (a,'->',c);
     hanoi (n-1,b,a,c)
    end
  end;
 begin
  write('圆盘数:');
  read(n);
  writeln('移动步骤如下:');
  hanoi(n,'A','B','C')
 end.
```

[运行实例]

```
圆盘数:3
移动步骤如下:
A->C
A->B
C->B
A->C
B->A
```

B->C

A->C

7.5.2 间接递归

以下两种情况都将引起子程序的间接递归调用：

① 在具有嵌套关系的子程序中，内层子程序调用外层子程序；

② 在作了超前引用说明的具有并列关系的子程序中，各子程序循环调用（例如，A调用B，B调用A；或A调用B，B调用C，C调用A；等等）。

什么是超前引用说明？

假设主程序要调用子程序A和子程序B，子程序A和子程序B也要互相调用。这时，如果采用嵌套结构（例如，在A的说明部分定义B，即A嵌套B），则主程序就无法调用处于内层的那个子程序；如果采用并列结构（例如，先定义A，后定义B），则在先定义的子程序中将出现对尚未定义的子程序的调用，违背了子程序必须先定义后使用的原则。解决这个矛盾的方法是采用超前引用说明。所谓超前引用说明是指在完整定义一个子程序之前，先写出其首部，并用预定义标识符 forward 标识之。具体用法，请看以下实例。

[例7.12] 在下面这个不完整的程序中，有两个并列的子程序SPA和SPB。主程序可以调用这两个子程序；两个子程序之间也可以相互调用。

[程序a]

```
program ch712a;
 var i:integer;
 function SPB(y:integer):integer;
   forward;   { 对SPB作超前引用说明 }
 procedure SPA(x:integer);
  var j:real;
  begin          { SPA的执行部分 }
   …
   j:=SPB(x); { 调用SPB }
   …
  end;
 function SPB;
  var k:real;
```

```
  begin          { SPB的执行部分 }
   …
   SPA(y);       { 调用SPA }
   …
  end;
 begin           { 主程序的执行部分 }
  …
  i:=SPB(i);     { 调用SPB }
  …
  SPA(i);        { 调用SPA }
  …
 end.
```

在定义SPA之前，先作关于SPB的超前引用说明，即写出函数SPB的首部、用预定义标识符 forward 代替函数体（注意：forward 前后有分号）。对SPB的实际定义放在过程SPA的定义之后，在函数体的前面只写出保留字 function 和函数名（省略函数名后面的括号、括号内的参数表、括号后的函数值类型）。

也可以采用另外一种形式，即对SPA作超前引用说明：

[程序b]

```
program ch712b;
 var i:integer;
 procedure SPA(x:integer);
   forward;    { 对SPA作超前引用说明 }
 function SPB(y:integer):integer;
  var k:real;
  begin          { SPB的执行部分 }
   …
   SPA(y);       { 调用SPA }
   …
  end;
 procedure SPA;{ 注意：过程名后的括号、括号内的参数表必须省略 }
  var j:real;
  begin          { SPA的执行部分 }
   …
```

```
    j:=SPB(x); { 调用SPB }
    …
  end;
begin          { 主程序的执行部分 }
  …
  i:=SPB(i);  { 调用SPB }
  …
  SPA(i);      { 调用SPA }
  …
end.
```

[例7.13] 输入一个以字符串形式给出的表达式，计算并输出这个表达式的值.

这里只讨论较简单的表达式：允许带括号；运算对象都是无符号整数；运算符只有加（+）、减（-）、乘（*）、除（/）4种，其中运算符“/”相当于保留字div．另外，还假定输入的表达式是正确的，没有语法错误．图7.7给出了这种表达式的语法．

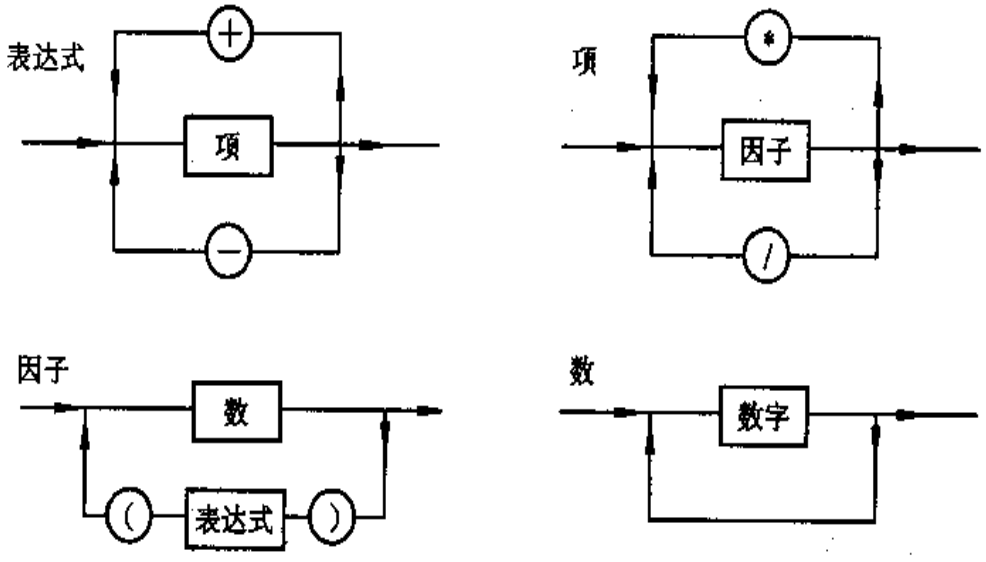


图7.7 表达式的语法

[解题思路]

根据语法图对这个复杂的程序设计任务进行分解，从主程序开始设计．

如果能够定义过程子程序

```
expression(ch,value)
```

（其功能是计算出以字符ch 开头的表达式的值，将这个值存入变量value

中，将紧跟在表达式后面的那个字符存入变量ch中）则主程序的执行部分将是非常的简单：

```
begin
 write('输入表达式:');
 read(ch); { 把第一个字符读到变量ch中 }
 expression(ch,value);          { 计算 }
 write('计算结果:',value)       { 输出 }
end.
```

由表达式的语法图可知，一个表达式至少包含一个项，但在第一个项的后面可以有也可以没有更多的项．因此，如果能够定义过程子程序

term(ch,value)

（其功能是计算出以字符ch开头的项的值，将这个值存入变量value中，将紧跟在项后面的那个字符存入变量ch中）则使用while结构很容易写出expression过程的定义：

```
procedure expression(var ch:char;
                     var value:integer);
 var
  op:char;
  nextvalue:integer;
 begin
  term(ch,value);          { 计算第一个项 }
  while (ch='+') or (ch='-') do
   begin
    op:=ch;                { 保存运算符 }
    read(ch);
    term(ch,nextvalue); { 计算下一个项 }
    if op='+'
    then value:=value+nextvalue
    else value:=value-nextvalue
   end
 end;
```

由于项的语法图和表达式的语法图非常相似，故可以仿照expression过程，写出term过程的定义。在下面给出的过程定义中，子程序

factor(ch,value)

的功能是计算出以字符ch 开头的因子的值，将这个值存入变量value中，将紧跟在因子后面的那个字符存入变量ch中。

```
procedure term (var ch:char;
                var value:integer);
 var   op:char;
       nextvalue:integer;
 begin
  factor(ch,value);            { 计算第一个因子 }
  while (ch='*')or(ch='/') do
   begin
    op:=ch;                    { 保存运算符 }
    read(ch);
    factor(ch,nextvalue);      { 计算下一个因子 }
    if op='*'
    then value:=value*nextvalue
    else value:=value div nextvalue
   end
 end;
```

因子或者以数字开头，或者以左括号开头。可以用 if-then-else 结构实现factor过程：

```
procedure factor(var ch:char;
                 var value:integer);
 begin
  if (ch>='0')and(ch<='9')
  then number(ch,value)
  else { ch='(' }
   begin
    read(ch);
    expression(ch,value);
    read(ch){读入紧跟')'后面的那个字符}
   end
 end;
```

在factor过程中调用了将数字串转换成整数的过程number(ch,value)，它的功能是把以字符ch 开头的数字串转换成整数并存入变量value中，将紧跟

在数字串后面的那个字符存入变量ch中.

下面给出number过程的定义.

```
procedure number(var ch:char;
                 var value:integer);
 begin
  value:=0;
  repeat
   value:=10*value+(ord(ch)-ord('0'));
   read(ch)
  untile (ch<'0')or(ch>'9')
 end;
```

至此，我们已定义了4个子程序. 由于主程序和子程序之间存在如图7.8所示的调用关系，因此，程序可以采用嵌套结构:

- 在主程序的说明部分定义expression;
- 在expression的说明部分定义term;
- 在term的说明部分定义factor;
- 在factor的说明部分定义number.

也可以采用并列结构:

- 对expression作超前引用说明;
- 依次定义number、factor、term和expression.

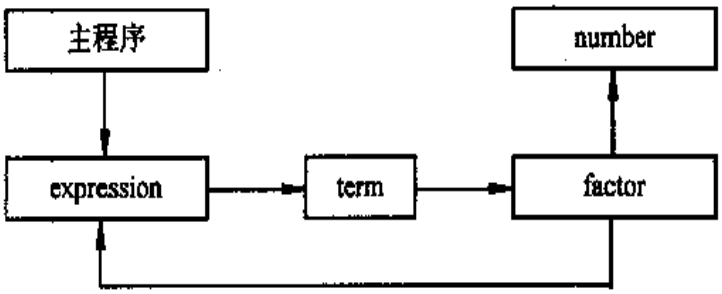


图7.8　主程序和子程序之间的调用关系

[程序a]

```
program ch713a;
{采用嵌套结构}
 var ch:char;
     value:integer;
 procedure expression(var ch:char;
```

```
                        var value:integer);
var op:char;
    nextvalue:integer;
precedure term (var ch:char;
                    var value:integer);
 var op:char;
     nextvalue:integer;
 precedure factor (var ch:char;
                      var value:integer);
  procedure number (var ch:char;
                       var value:integer);
   begin {number的执行部分}
    value:=0;
    repeat
     value:=10*value+(ord(ch)-ord('0'));
     read(ch)
    until (ch<'0')or(ch>'9')
   end;
  begin { factor的执行部分 }
   if (ch>='0')and(ch<='9')
   then number(ch,value)
   else { ch='(' }
    begin
     read(ch);
     expression(ch,value);
     read(ch) { 读入紧跟')'后面的那个字符 }
    end
 end;
begin { term的执行部分 }
 factor(ch,value);          { 计算第一个因子 }
 while (ch='*')or(ch='/') do
  begin
   op:=ch;                  { 保存运算符 }
```

```
        read(ch);
        factor(ch,nextvalue);        { 计算下一个因子 }
        if op='*'
        then value:=value*nextvalue
        else value:=value div nextvalue
       end
    end;
  begin { expression的执行部分 }
    term(ch,value);              { 计算第一个项 }
    while (ch='+')or(ch='-') do
      begin
        op:=ch;                  { 保存运算符 }
        read(ch);
        term(ch,nextvalue); { 计算下一个项 }
        if op='+'
        then value:=value+nextvalue
        else value:=value-nextvalue
      end
    end;
  begin { 主程序的执行部分 }
    write ('输入表达式:');
    read(ch); { 把第一个字符读到变量ch中 }
    expression(ch,value);          { 计算 }
    write('计算结果:',value)       { 输出 }
  end.
```

[程序b]

```
program ch713b;
{采用并列结构}
  var ch:char;
      value:integer;
  procedure expression (var ch:char;
                        var value:integer);
    forward;        { 对expression作超前引用说明 }
```

```
procedure number(var ch:char;
                 var value:integer);
 begin
  value:=0;
  repeat
   value:=10*value+(ord(ch)-ord('0'));
   read(ch)
  until (ch<'0')or(ch>'9')
 end;

precedure factor (var ch:char;
                  var value:integer);
 begin
  if (ch>='0')and(ch<='9')
  then number(ch,value)
  else { ch='(' }
   begin
    read(ch);
    expression(ch,value);
    read(ch) { 读入紧跟')'后面的那个字符 }
   end
 end;

precedure term (var ch:char;
                var value:integer);
 var op:char;
     nextvalue:integer;
 begin
  factor(ch,value);          { 计算第一个因子 }
  while (ch='*') or (ch='/') do
   begin
    op:=ch;                  { 保存运算符 }
    read(ch);
    factor(ch,nextvalue);    { 计算下一个因子 }
    if op='*'
```

```
      then value:=value*nextvalue
      else value:=value div nextvalue
    end
  end;

procedure expression;
 var op:char;
     nextvalue:integer;
 begin
  term (ch,value);          { 计算第一个项 }
  while (ch='+')or(ch='-') do
   begin
    op:=ch;                 { 保存运算符 }
    read(ch);
    term(ch,nextvalue); { 计算下一个项 }
    if op='+'
    then value:=value+nextvalue
    else value:=value-nextvalue
   end
 end;

begin { 主程序的执行部分 }
 write('输入表达式:');
 read(ch); { 把第一个字符读到变量ch中 }
 expression(ch,value);          { 计算 }
 write('计算结果:',value)       { 输出 }
end.
```

习 题 七

7.1 填空题。

(1) _____的程序段称为子程序。

(2) 子程序有_____和_____两种，分别用保留字_____和_____开头。

(3) 在定义函数子程序时，函数体中至少要有一个_____语句。

(4) 在表达式中只能出现_____调用，不能出现_____调用。

(5) 实参与形参必须按序一一对应，_____相等，_____兼容。

(6) 过程体与函数体的唯一区别在于_____。

(7) 与数值形参对应的实参可以是_____，与变量形参对应的实参只能是_____。

(8) 在两个子程序之间传递数据可以用_____，还可以用_____。

(9) 如果某个参数只用于输入子程序所要处理的数据，则该参数一般用_____形参。

(10) 如果某个参数要用于输出子程序的处理结果,则该参数只能用_____形参。

(11) 数组作为参数时，一般都用_____形参。

(12) 在子程序和子程序之间，存在_____和_____两种关系。

(13) 直接递归是指_____。

(14) 间接递归是指_____。

(15) 全程变量的作用域是_____，局部变量的作用域是_____。

7.2 设有变量说明及函数定义：

```
var a:integer; b:real;
function d(e:real; var f:integer):real;
 begin
  d:=e+f;
  f:=f*f
 end;
```

请指出下列哪些语句是对的,哪些语句是错的。

(1) write(d(1,a))

(2) a:=d(2.5,b)

(3) write(d(1,2*b))

(4) a:=d(a,a)

(5) write(d(d(1,a),a))

7.3 写出下列程序的输出结果：

(1)

```
var a:integer;
```

```
function f(x:integer):integer;
 begin
  f:=sqr(x)+12*x-5
 end;
begin
 a:=4;
 writeln(f(a));
 writeln(f(a+1))
end.
```

(2)

```
var a,x,y:integer;
procedure p1(x:integer;var y:integer);
 var a:integer;
 begin
  x:=x+123;
  y:=y-219;
  a:=x+y
 end;
begin
 a:=100;x:=120;y:=113;
 p1(a,x);writeln(x:6,y:6,a:6);
 p1(y,y);writeln(x:6,y:6,a:6);
 p1(y,x);writeln(x:6,y:6,a:6)
end.
```

(3)

```
var n:integer;
procedure p2(n:integer);
 var i:integer;
 begin
  if n>0 then begin
   p2(n-1);
   for i:=1 to n do write(n);
   writeln
  end
```

```
  end;
 begin
  n:=5;p2(n)
 end.
```

(4)

```
 var x:integer;
 procedure p3(x:integer);
  var r:0..7;
  begin
   r:=x mod 8;
   if x<>0 then p3(x div 8);
   write(r)
  end;
 begin
  x:=586;p3(x)
 end.
```

7.4 写一个自定义函数digit(n,k),其函数值等于正整数n的右起第k位数字。例如，

digit(254693,2)=9

digit(7622,5)=0

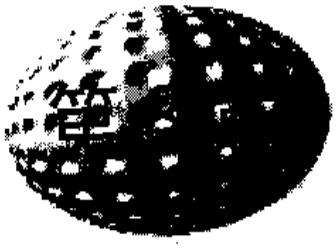
7.5 当x>1时，Hermite多项式$H_n(x)$定义如下：

$H_0(x)=1$

$H_1(x)=2x$

$H_n(x)=2xH_{n-1}(x)-2(n-1)H_{n-2}(x)$

试分别写出计算Hermite多项式$H_n(x)$值的递归函数和非递归函数。



集合和记录

集合类型和记录类型都属于构造类型。虽然它们的应用不如数组类型那样普遍,但可以使某些问题的描述更为直观、合理，处理起来更为方便。

8.1 集　合

和数组一样，集合也是由一些性质相同的元素构成的，所不同的是，集合中元素的个数是可变的，而且集合变量只作为一个整体来使用，不能单独使用其元素。

8.1.1 定义集合类型

定义集合类型的格式为

```
集合类型名=set of 元素类型
```

其中，set和of都是保留字，集合的元素类型可以是字符类型、枚举类型或子界类型。

一个集合变量所包含的元素个数由元素类型决定，但最多只能有256个元素。

[例8.1] 定义3种集合类型，分别表示大写字母集合、每周工作日集合、家庭成员年龄集合。

```
type
  letter='A'..'Z';
  weekday=(sun,mon,tue,wen,thu,fri,sat);
  age=0..100;
```

```
charset=set of letter;
dayset=set of weekday;
ageset=set of age;
```

集合变量的说明与其它变量的说明相同。例如，

```
var
 ch:charset;
 workday:dayset;
 family:ageset;
```

当然，也可以将集合类型的定义并入集合变量的说明中。例如，

```
var
 family: set of 0..100;
```

8.1.2 集合的值

集合的值是一个用方括号括起来的元素序列，元素之间用逗号分隔。例如，

```
['A', 'E', 'I', 'O', 'U', 'Y']
[sun,sat]
```

集合的值与方括号内元素的排列顺序无关。因此，[sun,sat]和[sat,sun]代表同一个值。

在方括号内某个元素重复出现，对集合的值没有影响。因此，[sun,sun,sat]和[sun,sat]代表同一个值。

如果出现在方括号内的元素是连续的，则可采用简化表示法（用符号“..”）。例如，[1,2,3,4,5,7,8,9,10,15]，可以表示成[1..5,7..10,15]。

方括号内可以没有元素，这样的集合称为空集。任何集合类型的空集都是相同的，用[ ]表示。

包含了所有元素的集合称为全集。例如，在例8.1中，集合类型charset的元素取值为'A'..'Z'，这种类型的全集为['A'..'Z']。

8.1.3 集合的并、交、差运算

两个类型相同的集合之间可以进行并、交、差运算。

1. 并运算

并运算的运算符用“+”号表示。两个集合的并运算的结果是个集合，它包含了这两个集合中的所有元素。例如，表达式 [sun,sat]+[sat,fri] 的值为 [sun, sat, fri]。

2. 交运算

交运算的运算符用“*”号表示。两个集合的交运算的结果是个集合，它包含这两个集合中的共有元素。例如，表达式 [sun,sat]*[sat,fri,thu] 的值为 [sat]；表达式 [sun,sat]*[mon,fri] 的值为[ ]。

3. 差运算

差运算的运算符用“-”号表示。两个集合的差运算的结果是个集合，它由那些只在第一个集合中出现而不在第二个集合中出现的元素构成。例如，表达式 [sun,sat]-[sat,fri] 的值为 [sun]；表达式 [mon]-[mon,tue] 的值为[ ]。

8.1.4 集合的关系运算和包含运算

1. in运算

关键字in是关系运算符，用来检测某个数据是不是集合元素。对于关系式 c in b，如果集合b中有元素c，则结果为true，否则结果为 false。例如，若枚举变量day的值为 sun或sat，则表达式 day in [sun,sat] 的值为true，否则为 false。

2. 比较运算

两个集合可以进行相等或不相等的比较运算。例如，下列关系运算的结果为true：

```
[sun,sat]=[sat,sun]
[sun,sun,sat]=[sun,sat]
```

但是，被比较的两个对象必须是同类型的集合。因此，下面两个表达式都是错误的：

```
[sun,sat]=[0,6]    {两个集合的元素类型不同}
[sun]=sun          {一个是集合,另一个是枚举值}
```

3. 包含运算

包含运算的运算符用“<=”或“>=”表示．对于两个类型相同的集合a和b，如果b集合是a集合的一个子集，即集合a中包含了集合b中的所有元素，则包含运算

a>=b {读作：a包含b}

和

b<=a {读作：b包含于a}

的结果为true，否则为false．例如，下列包含运算的结果为true：

[1,3,5]<=[1..5]

[4,7,8]>=[ ]

但是，参与包含运算的两个对象必须是同类型的集合．因此，下面两个表达式是错误的：

[mon,fri]>=[1,3,5]

[1,3,5]>=3

8.1.5 集合的输入与输出

给集合变量设置初始值，不能用read语句，只能用赋值语句．例如，

```
workday:=[mon..fri];
ch:=[ ];
```

为了输出一个集合变量的值，不能直接用write语句，可以采用类似于输出枚举值的方法．例如，设有下述定义与说明：

```
type
 weekday=(sun,mon,tue,wen,thu,fri,sat);
 dayset=set of weekday;
const
 daynames: array [weekday] of string[3]
          =('sun', 'mon', 'tue', 'wen', 'thu', 'fri', 'sat');
var
 workday:dayset;
     day:weekday;
 separator:char;
```

可用下面的程序段直观地输出集合变量workday的值：

```
write('[');
separator:=' ';
for day:=sun to sat do
 if day in workday
 then begin
        write(separator,daynames[day]);
        separator:=','
      end;
write(' ]')
```

8.1.6 定义集合常量

定义集合常量的一般形式为

```
集合常量名:集合类型=集合值
```

例如：

```
type
 letter=set of 'A'..'Z';
const
 vowels:letter=['A','E','I','O','U'];
 hexdigits: set of char=['0'..'9','A'..'Z','a'..'z'];
```

集合常量在编程实践中用得较少，这是因为集合值本来就不能用read语句输入，只能通过赋值得到，即使用集合常量不会给编程带来多少方便。

8.1.7 应用举例

[例8.2] 输入一个以句号结尾的英文句子，分别统计句中元音字母和辅音字母出现的次数。

[解题思路]

通过赋值产生元音字母的集合us和辅音字母的集合cs；对读入的字符c进行检测：如果c在us中，则元音字母个数un加1；如果c在cs中，则辅音字母个数cn加1。

[程序]

```
program ch82;
 type letter=set of char;
```

```
var us,cs:letter;
      un,cn:integer;
          c:char;
begin
 us:=['A','E','I','O','U'];
 cs:=['A'..'Z']-us;
 un:=0;cn:=0;
 writeln('请输入一个以句号结尾的句子:');
 read(c);
 while c<>'.' do
  begin
   c:=upcase(c);{如果读到的是小写字母,则把它转换成大写字母}
   if c in us    { 判别c是否为元音字母 }
   then un:=un+1;
   else if c in cs { 判别c是否为辅音字母 }
    then cn:=cn+1;
   read(c)
  end;
 writeln('统计结果:');
 writeln('元音字母出现',un,'次');
 writeln('辅音字母出现',cn,'次')
end.
```

[运行实例]

```
请输入一个以句号结尾的句子:
Happy new year to you.
统计结果:
元音字母出现7次
辅音字母出现10次
```

程序中使用了一个新的标准函数upcase，它只对小写字母字符起作用：把小写字母字符转换成对应的大写字母字符.

在这个程序中，如果不使用集合变量和集合运算，则判别c是否为元音字母就得用条件

```
(c='A')or(c='E')or(c='I')or(c='O')or(c='U')
```

显然是很麻烦的.

[例8.3]　列表输出从2到100之间的所有素数。

[解题思路]

首先,将集合p初始化为一个空集,将集合s初始化为一个全集。然后，反复执行下列操作:选取s中最小的数k;把k放入p中;从s中删去k及其所有的倍数。当s变为空集时,所有的素数已在集合p中。

上述求素数的方法称为筛选法.

[程序]

```
program ch83;
 const n=100;
 type pset=set of 2..n;
 var a:pset;

 procedure getprimary (var p:pset);
  { 产生素数集合p }
  var k:integer;
  procedure delset (k:integer; var s:pset);
   { 从集合s中删去k及其所有的倍数}
   var i:integer;
   begin
    i:= k;
    while i<=n do
     begin
      s:=s-[i];    {从集合s中删去元素i}
      i:=i+k
     end
   end;
  {getprimary的执行部分}
  begin
   p:=[ ];s:=[2..n];
   k:=1;
   repeat
    repeat k:=k+1
    until k in s;     {选取s中最小的数k}
    p:=p+[k];         {k放入p中}
    delset(k,s);      {从s中删去k及其所有的倍数}
```

```
    until s=[ ]        {直到s变为空集}
  end;

 procedure putprimary(p:pset);
  { 列表输出集合p中的元素 }
  const m=5; { 每行5个元素 }
  var i,k:integer;
  begin
   writeln ('素数表:');
   k:=0;
   for i:=2 to n do
    if i in p then
     begin
      write (i:4);
      k:=k+1;
      if k mod m=0 then writeln
     end
  end;
 {主程序的执行部分}
 begin
  getprimary(p);
  putprimary(p)
 end.
```

[运行结果]

素数表:

```
 2   3   5   7  11
13  17  19  23  29
31  37  41  43  47
53  59  61  67  71
73  79  83  89  97
```

[例8.4] 邮局要发行一套有4种不同票面值的邮票，如果规定每封信最多只能贴3张邮票，则存在整数R，使得用不超过3张的邮票，就可以贴出连续的整数邮资1~R．写一个程序，找出这4种邮票的面值数，使得R值最大．

[解题思路]

显然,只有4种邮票面值互不相同，才有可能得到最大的R值。

设4种邮票的面值为a<b<c<d．显然，a只能取1；b的可能取值在a+1到3a+1之间；c的可能取值在b+1到3b+1之间；d的可能取值在c+1到3c+1之间．

对a,b,c,d的每一种组合，分别计算满足题目要求的R值，其中使R取最大值的那一种组合，即是所求结果．

可以通过一个四重循环来确定由某个组合a,b,c,d可能得到的R值．设面值为a的邮票可以贴n1张(0≤n1≤3)，面值为b的邮票可以贴n2张(0≤n2≤3-n1)，面值为c的邮票可以贴n3张(0≤n3≤3-n1-n2)，面值为d的邮票可以贴n4张(0≤n4≤3-n1-n2-n3)．对满足条件

n1+n2+n3+n4≤3

的每一组n1,n2,n3,n4，计算积之和

sum=n1*a+n2*b+n3*c+n4*d

并将sum加入到集合s中(s初始值为空集)．最后，检测1,2,3,…,n是否在集合s中，若n在s中，而n+1不在s中，则R=n．

[程序]

```
program ch84;
 const m=3;{信封上最多可贴m张邮票}
 var a,b,c,d:integer;
     R,Q,x1,x2,x3,x4:integer;
 function number(a,b,c,d:integer):integer;
 {确定由面值为a,b,c,d的4种邮票可得到的R值}
  var n1,n2,n3,n4,sum,n:integer;
      s:set of 1..100;
  begin
   s:=[ ];
   for n1:=0 to m do
    for n2:=0 to m-n1 do
     for n3:=0 to m-n1-n2 do
      for n4:=0 to m-n1-n2-n3 do
       if n1+n2+n3+n4<=m {不超过m张}
       then begin
        sum:=n1*a+n2*b+n3*c+n4*d;{得到一种邮资}
        s:=s+[sum]                    {放入s集合}
       end;
   n:=1;
```

```
    while n in s do n:=n+1;
    number:=n-1                          {返回最大的R值}
  end;
 begin
  a:=1; R:=0;
  for b:=a+1 to m*a+1 do
   for c:=b+1 to m*b+1 do
    for d:=c+1 to m*c+1 do
    begin
     Q:=number(a,b,c,d);
     if Q>R then
     begin
      R:=Q;
      x1:=a;x2:=b;x3:=c;x4:=d
     end
    end;
  writeln;
  write('4种票面值为:',x1:4,x2:4,x3:4,x4:4)
 end.
```

[运行结果]

4种票面值为: 1 4 7 8

[例8.5] 士兵排队问题。设有一队士兵，人数不超过26个，不知道每个士兵的具体身高，只知道相互之间的高矮关系（例如：A比C高，C比B高，等等）。请写一个程序，输入士兵人数及相互之间的高矮关系，将他们按从高到矮的顺序排成一排。

[解题思路]

输入士兵人数n，用字母'A','B',…,chr(ord('A')+n-1)表示这些士兵。

用二维数组r表示士兵之间的高矮关系：若A比B高，则r['A','B']='1'，否则r['A','B']='0'。

以字符串形式输入士兵之间的高矮关系。例如，若A比B高，则输入字符串'AB'，将'A'和'B'加入集合s（初值为空），将r['A','B']置'1'（r初值为全'0'）。

所有的高矮关系都输入以后，如果集合s不等于['A'..chr(ord('A')+n-1)]，则说明输入数据有误，否则进行排序处理。

逐列检测r数组，若第c列是数组中仅有的全由零元素组成的列，则说

明c是最高的。在输出c，并且把第c列元素全置成'1'，把第c行元素全置成'0'后，重新检测数组r。

在检测过程中，如果发现没有全零的列或者全零的列不止一列，则说明输入的关系不完备或者矛盾。

[程序]

```
program ch85;
 type arr=array['A'..'Z','A'..'Z']of '0'..'1';
 var r:arr;
      n:integer;
  mark:boolean;
 procedure getdata (var r:arr;
                    var n:integer;
                    var mark:boolean);
{建立n(1<n<27)个士兵之间的高矮关系r,mark标记输入是否正确}
 var s,s0: set of char;
     a,b,u:char;
     ab: string[2];
     finish: boolean;
 begin
  repeat
   write('输入士兵人数:');
   read(n)
  until (n>1) and (n<27);
  u:=chr(ord('A')+n-1);
  s0:=['A'..u]; {全体士兵的集合}
  {r数组初始化}
  for a:='A' to u do
   for b:='A' to u do r[a,b]:='0';
  s:=[ ];
  finish:=false;
  writeln('输入高矮关系:');
  repeat
   readln(ab);
   a:=upcase(ab[1]);
```

```
    b:=upcase(ab[2]);
    if not (a in s0)or not (b in s0)
    then finish:=true
    else begin
           r[a,b]:='1';
           s:=s+[a,b] {集合s中加入元素a和b}
         end
  until finish;
  mark:=s=s0    {给变量形参mark赋值}
 end;
procedure sorting (var r:arr;
                   var n:integer);
{根据n个士兵之间的高矮关系进行排序}
{n返回不能确定高矮关系的士兵人数}
 var a,b,c,u:char;
     i:integer;
 begin
  u:=chr(ord('A')+n-1);
  repeat
   i:=0; {对r中全'0'的列进行计数}
   b:='A'; {从第'A'列开始检测}
   repeat
    a:='A';
    while (a<=u)and(r[a,b]='0')
     do a:=succ(a);
    if a>u  {如果这一列全'0'}
    then begin
     c:=b;    {c标记全'0'列}
     i:=i+1   {全'0'列的列数加1}
    end;
    if i<=1 then b:=succ(b)  {准备检测下一列}
   until (b>u)  {直到每一列都检测过了}
       or (i>1); {或发现有多个全'0'列为止}
   if i=1 {如果全'0'列只有一列}
```

```
      then begin
        write(c); {输出一个士兵的标识}
        n:=n-1;    {剩余士兵人数减1}
        for a:='A' to u do
        begin
          r[a,c]:='1'; {第c列全置'1'}
          r[c,a]:='0'   {第c行全置'0'}
        end
      end
    until (n=0)   {直到输出了n个士兵的标识}
      or (i<>1) {或发现有多个全'0'列为止}
  end;
 begin
  getdata(r,n,mark);   {输入}
  write('处理结果:');
  if mark   {如果输入是正确的}
  then begin
         sorting(r,n); {排序}
         if n>0 {有些士兵的高矮关系不能确定}
         then writeln('关系不完备或矛盾!')
       end
  else write('输入数据错!')
 end.
```

[运行实例]

```
输入士兵人数:5
输入高矮关系:
cd
bd
ac
bc
ab
de
be
00
```

处理结果:ABCDE

8.2 记 录

记录是由一组相关的元素构成的，这些元素的类型可以不相同（这是记录和数组的主要区别）。

构成记录的元素习惯上称为“域”或“字段”。

8.2.1 定义记录类型

定义记录类型的格式为

```
记录类型名=record
  域名1:类型1;
  域名2:类型2;
  …
  域名n:类型n
        end
```

其中，record和end 是保留字；域名（即元素名）是用户定义的标识符；每个域的类型可以是包括记录类型在内的任何一种类型。

[例8.6] 一张记载某个职工基本情况的卡片上列有工号、姓名、性别、出生日期、住址等信息。对于这张卡片，可用记录类型emp来描述：

```
type
 sexs=(female,male);
 dates=record
         year:1900..2000;
         month:1..12;
         day:1..31
       end;
emp=record
      no:integer;
      name:string[16];
      sex:sexs;
      birthdate:dates;
```

```
        adds:string[20];
      end;
```

emp记录类型包含5个域。其中，birthdate域也是个记录，它有year、month、day三个域。

记录变量的说明与其它变量的说明相同。例如，

```
var
  person:emp;
  group:arrary[1..n] of emp;
```

这里，person是个记录类型的变量，可用来存放一个职工的信息；group是个元素为记录类型的一维数组，可用来存放n个职工的信息。

8.2.2 使用记录变量

在大多数情况下，我们只能通过引用记录变量中的域来使用记录变量。引用记录变量中某个域的方法是：写出记录变量名和域名，在记录变量名和域名之间加一个小圆点'.'。例如，可用下列方法引用记录变量person中的各个域：

```
person.no          { 表示某职工的编号     }
person.name        { 表示该职工的姓名     }
person.sex         { 表示该职工的性别     }
person.birthdate   { 表示该职工的出生日期 }
person.adds        { 表示该职工的住址     }
```

其中，person.birthdate仍是个记录，引用其中的3个域要用如下形式：

```
person.birthdate.year    { 表示该职工的出生年份       }
person.birthdate.month   { 表示该职工的生日是在哪一月 }
person.birthdate.day     { 表示该职工的生日是在哪一天 }
```

对记录变量中每个域可以进行哪些运算取决于该域的类型。

[例8.7] 写一个程序段，读入一个职工的信息。

设ch是个字符型变量，可以写出如下程序段：

```
read(person.no,person.name);
read(ch);
read(person.birthdate.year,person.birthdate.month,person.birthdate.day);
read(person.adds)
case upcase(ch) of
```

```
'F':person.sex:=female;
'M':person.sex:=male
end
```

emp类型中第3个域sex属于枚举类型，不能直接用 read语句输入，在例8.7中通过读入一个字符进行间接赋值。

对于两个类型相同的记录，可以进行整体的赋值。例如，赋值语句

```
group[1]:=person
```

是合法的，其结果是将person中每个域的值赋给group[1]中同名的域。

8.2.3 开域语句

开域语句有两种格式。

格式1：

```
with 记录名 do 语句
```

格式2：

```
with 记录名1,记录名2,…, do 语句
```

其中，with和do是保留字；with后面的记录名是要打开的记录，这些记录名可以是记录类型的变量名或域名。对于第二种形式，记录名1后面的记录名都必须是那些内嵌在记录名1中的记录类型的域名。do后面的语句可以是一个简单语句，也可以是复合语句。在这个语句中，引用开域记录中的域名时可以只写域名而不必写记录名和小圆点。

使用开域语句的目的是为了简化对记录域的引用。例如，利用开域语句，可以将例8.7中读入职工信息的程序段简化为

```
with person do
 begin
  read(no,name);
  read(ch);
  with birthdate do read(year,month,day);
  read(adds)
  case upcase(ch) of
   'F':sex:=female;
   'M':sex:=male
  end
```

```
    end
```

或者

```
  with person,birthdate do
   begin
    read(no,name);
    read(ch);
    read(year,month,day);
    read(adds)
    case upcase(ch) of
     'F':sex:=female;
     'M':sex:=male
    end
   end
```

8.2.4　定义记录常量

定义记录常量的格式为

```
记录常量名:记录类型=(域名1:值1;
                      域名2:值2;
                      ...
                      域名n:值n)
```

其中，域名的排列顺序必须和类型定义中的顺序相同．例如，

```
type
 dates=record
        year:1900..2000;
        month:1..12;
        day:1..31
       end;
const
 lastday:dates=(year:1999;month:12;day:31);
```

和集合常量一样，记录常量在编程实践中也用得不多．

8.2.5 带变体的记录

许多人事部门使用的“职工基本情况登记卡”中都设有“婚姻状况”的栏目，对已婚职工要求填写对方姓名、出生日期、工作单位等内容，而对未婚职工则不填写这些内容。

对于这种卡片，可用带变体的记录类型emp来描述：

```
type
 sexs=(female,male);
 marsta=(married,single);
 dates=record
  year :1900..2000;
  month:1..12;
  day   :1..31
        end;
emp=record
 no:integer;
 name:string[16];
 sex:sexs;
 birthdate:dates;
 adds:string[20];
 case ms:marsta of
 married:  {对已婚者还有3个域}
 ( spname:string[16];
    spbirth:dates;
    spwn:string[20]
 );
 single: {对未婚者没有其它的域}
 ()
     end;
```

在这个emp记录类型定义中，位于record和 case之间的那部分（即前5个域）构成记录的固定部分；由case引出的那部分（从case到end）构成记录的变体部分．其中，ms称为标记域．对于任何一个emp类型的变量来说，除了有no、name、sex、birthdate、adds这5个固定部分的域和标记域ms外，

是否还有其它的域，取决于ms的值：如果ms的值为married，则除了有上述6个域以外，还有spname、spbirth和spwn这3个域；如果ms的值为single，则除了有上述6个域以外，没有其它的域。

定义带变体记录类型的一般格式为

```
记录类型名=record
  域名1:类型1;
  域名2:类型2;
  …
  域名n:类型n;
  case 标记域:标记域类型 of
   标号1:(域表1);
   标号2:(域表2);
   …
   标号m:(域表m)
          end;
```

其中，标记域类型只能是有序类型；标号1至标号m是该有序类型的取值；每个域表要用圆括号括起来，域表的形式和记录的固定部分的形式相同，每个域表中可以包含一个域、几个域，也可以一个域都不包含（是个空域表）。

对于一个带变体的记录变量（即用带变体的记录类型说明的变量）来说，其固定部分域和标记域总是可访问的，而变体部分哪些域可访问，哪些域不可访问，则要由标记域的值来决定。

例如，假设p是emp类型的记录变量，如果p.ms的值为single，则只能访问6个域：

p.no

p.name

p.sex

p.birthdate

p.adds

p.ms

如果p.ms的值为married，则除了可以访问上述6个域之外，还可以访问以下3个域：

p.spname

p.spbirth

p.spwu

8.2.6 应用举例

[例8.8] 输入多边形各端点的坐标值，计算并输出多边形的周长。

[解题思路]

将各端点的坐标存入记录数组p中，通过一个循环来计算相邻两点p[i]和p[i+1]之间的距离。

[程序]

```
program ch88;
 const maxn=100;
 type coordinate=record
          x,y:real
      end;
 var p:array[0..maxn]of coordinate;
      s:real; {表示周长}
      i,n:integer;
 function len(a,b:coordinate):real;
 { 计算a,b两点间的距离 }
  begin
   len:=sqrt(sqr(a.x-b.x)+sqr(a.y-b.y))
  end;
begin
 write('输入端点个数n:');
 read(n);
 writeln('顺序输入',n,'个端点的坐标值(x,y):');
 {将n个端点的坐标值存入p[0]至p[n-1]中}
 for i:=0 to n-1 do
 begin
  write(i+1,':');
  read(p[i].x,p[i].y);
 end;
 {累加n条边的长度}
```

```
  s:=0;
  for i:=0 to n-1 do
   s:=s+len(p[i],p[(i+1)mod(n+1)]);
  writeln('周长为:',s:6:3)
 end.
```

[运行实例]

输入端点个数n:4

顺序输入4个端点的坐标值(x,y):

1:　0　　2

2:　2　　0

3:　1　　-2

4:　-1　　0

周长为: 8.893

[例8.9]　设有3种几何图形：三角形、矩形、圆．已知三角形的三条边长，矩形的长和宽，圆的半径．写一个程序，其功能是：输入一系列几何图形的已知数据，找出其中面积最大的几何图形。

[解题思路]

几何图形的种类用枚举类型描述：

```
  fig=(triangle,rectangle,circle);
```

3种几何图形用同一种带变体的记录类型描述：

```
  figure=record
          area:real;
   case figes:fig of
    triangle :( a,b,c:real );
    rectangle:( l,w:real );
    circle    :( r:real );
          end
```

其中，a,b,c表示三角形的3条边长；l和w表示矩形的长和宽；r表示圆的半径。

输入一个字符('t','r','c'),将它转换成对应的枚举值；根据转换结果，按不同的计算公式计算各种几何图形的面积．

[程序]

```
program ch89;
 const pi=3.1415926;
```

```
type
 fig=(triangle,rectangle,circle);
 figure=record
          area:real;
  case figes:fig of
   triangle :( a,b,c:real );
   rectangle:( l,w:real );
   circle    :( r:real );
          end;
function s(a,b,c:real):real;
{根据3条边的长度计算三角形面积}
 var x:real;
 begin
  x:=(a+b+c)/2;
  s:=sqrt(x*(x-a)*(x-b)*(x-c))
 end;
var
 g,max:figure;{max表示面积最大的几何图形}
 c:char;
begin
 max.area:=0;
 repeat
  write('INPUT T,R,C:');
  readln(c); {输人一个表示几何图形种类的字符}
  c:=upcase(c);
  if c in ['T','R','C']  {输入正确}
  then begin
   {给标记域赋枚举值}
   case c of
   'T': g.figes:=triangle;
   'R': g.figes:=rectangle;
   'C': g.figes:=circle
   end;
   {计算面积}
```

```
    with g do
    case figes of
    triangle: {是三角形}
     begin
      readln(a,b,c);  {输入3条边的长度}
      area:=s(a,b,c)
     end;
    rectangle: {是矩形}
     begin
      readln(l,w);     {输入长和宽}
      area:=l*w
     end;
    circle: {是圆}
     begin
      readln(r);       {输入半径}
      area:=pi*r*r
     end
    end;
    {保存最大值}
    if g.area>max.area
    then max:=g
  end
until not (c in ['T','R','C']);
{输出}
write('面积最大的图形:');
with max do
case figes of
 triangle:
    write('三角形.边长:',a:8:2,b:8:2,c:8:2,' 面积:',area:8:2);
 rectangle:
    write('矩形.长和宽:',l:8:2,w:8:2,' 面积:',area:8:2);
 circle:
    write('圆.半径:',r:8:2,' 面积:',area:8:2)
end
```

```
 end.
```

[运行实例]

（略）

[例8.10]　输入整数n及n个无符号整数，将这n个无符号整数联成一个最大的多位整数并输出之．例如，将5个整数

6750　67　34　345　7

联成

767675034534

[解题思路]

无符号整数用integer类型表示，最大值为32767，即不超过5位．把输入的每个无符号整数都扩展成5位的字符串，新增数字都放在右边，与个位上的数字相同．例如，把上面的5个整数扩展成5个字符串：

'67500'，'67777'，'34444'，'34555'，'77777'

对这些字符串进行从大到小的排序：

'77777'，'67777'，'67500'，'34555'，'34444'

最后，依次输出各字符串所对应的无符号整数：

7，67，6750，345，34

用一维数组r存放所输入的数据，数组的元素是个记录，包含两个域：key域存放扩展后的字符串；value域存放输入的整数．

对数组r中的元素按key域的值排序，最后输出各元素value域的值．

[程序]

```
program ch810;
 const maxn=40;
 type
  node=record
    key:string[5];
    value:integer
       end;
 var i,j,k,n:integer;
      t:node;
      r:array[l,maxn]of node;
 begin
  write('输入n:')
  read(n);
```

```
    writeln('输入',n,'个无符号整数:');
    for i:=1 to n do with r[i] do
    begin
     read(value);      {读入一个整数}
     str(value,key);   {转换成字符串}
     j:=length(key);
     {按低位扩展,取左起5位为key值}
     key:=key+key[j]+key[j]+key[j]+key[j]
    end;
    { 排序 }
    for i:=1 to n-1 do
    begin
     k:=i;
     for j:=i+1 to n do
      if r[j].key>r[k].key {  比较大小 }
       then k:=j;
     if k>i
     then begin                { 移动记录 }
      t:=r[j];r[j]:=r[k];r[k]:=t
     end
    end;
    { 输出 }
    write('最大数字串:');
    for i:=1 to n do write(r[i].value)
  end.
```

[运行实例]

```
输入n:5
输入5个无符号整数:
6750    67    34    345    007
最大数字串:
767675034534
```

注意：由于r数组的元素是个记录，因此在进行排序时，比较的是记录的key域，而移动的是整个记录。

[例8.11]　某高校招收m名研究生（m<100），入学考试科目共5门：数

学、外语、政治、专业一、专业二。录取分数线为：总分在300分以上；各科成绩分别不低于58分、50分、55分、60分、60分。写一个程序，输入每个考生的考号、姓名、总分及各科成绩，按总分从高到低的顺序分别列表输出上线考生和未上线考生的信息（包括每个考生的考号、姓名、总分及各科成绩）。

[解题思路]

用记录类型描述一个考生的信息，用记录数组表示考生信息汇总表。将程序要完成的任务分解成若干个子任务，用子程序实现。

getdata(p) 用于读入一个考生的考号、姓名和5门课程的成绩，计算其总分，并作出是否上线的判断，将这些信息存入记录变量p中。

insert(p,s,n)用于把记录p插入到汇总表s中的适当位置。其中，n是汇总表中记录的个数。汇总表中的各记录按总分由高到低的顺序排列。

disp(s,n,mark)用于根据mark的值列出汇总表s中的记录：若mark=true，则列出上线考生的信息；若mark=false，则列出未上线考生的信息。

getlist(s,n)通过交替调用getdata(p)和insert(p,s,n)建立汇总表。

putlist(s,n)通过相继调用disp(s,n,true)和disp(s,n,false)列出上线考生信息和未上线考生信息。

[程序]

```
program ch811;
 const m=100;
 type number=1..m;
      stype=array[0..5]of integer;
      pr=record
           no:integer;       { 考号 }
           name:string[8]; { 姓名 }
           score:stype;      { 成绩 }
           tag:boolean       { 是否上线的标记 }
         end;
      list=array[number]of pr;
 const course:array[1..5]of string [6]
              =('数学','外语','政治','专业一','专业二');
       pass:stype
            =(300,58,50,55,60,60); { 录取分数线 }
 var s:list;
```

```
      n:integer; { 实际考生人数 }
procedure getdata(var p:pr);
 var j:integer;
 begin
  with p do
  begin
   write('考号:');readln(no);
   if no>0 then {考号不能为0}
   begin
    write('姓名:');readln(name);
    score[0]:=0;       { 总分置初值 }
    tag:=true;
    for j:=1 to 5 do
    begin
     write(course[j],':');   {显示科目名称}
     read(score[j]);         {输入分数}
     tag:=tag and (score[j]>=pass[j] ); {单科是否合格}
     score[0]:=score[0]+score[j]    {算总分}
    end;
    tag:=tag and(score[0]>=pass[0]) {不仅单科要合格,总分也要合格}
   end
  end
end;
procedure insert (p:pr;var s:list;var n:integer);
 var i:integer;
 begin
  i:=n; {从s[n]开始找插入位置}
  while(i>0)and(s[i].score[0]>p.score[0])do
  begin
   s[i+1]:=s[i]; {s[i]的值往后移}
   i:=i-1   {准备与前面一个比较}
  end;
  s[i+1]:=p; {插入}
```

```
    n:=n+1        {汇总表中的记录数加1}
  end;

procedure disp (var s:list;n:integer;
                mark:boolean);
  var i,j:integer;
  begin
    writeln;
    write('  考号 姓名          总分');
    for j:=1 to 5 do write(course[j]:10);
    writeln;
    for i:=1 to n do
      with s[i] do
      if tag=mark then
      begin
        write(no:4,name:8);
        for j:=0 to 5 do write(score[j]:10);
        writeln
      end
  end;

procedure getlist (var s:list; var n:integer);
  var i:integer;
      p:pr;
  begin
    writeln;
    n:=0;
    repeat
      getdata(p);
      if p.no>0 then insert(p,s,n)
    until eof or (p.no=0)
  end;

procedure putlist (var s:list;n:integer);
  var i:integer;
  begin
```

```
    writeln('上线考生:');
    disp(s,n,true);
    writeln('未上线考生:');
    disp(s,n,false)
  end;

begin
  getlist(s,n);
  putlist(s,n)
end.
```

[运行实例]

```
考号:98001
姓名:丁一
数学:70
外语:65
政治:68
专业一:80
专业二:85
考号:98002
姓名:王二
数学:60
外语:69
政治:62
专业一:78
专业二:89
考号:98004
姓名:李四
数学:58
外语:50
政治:70
专业一:68
专业二:80
考号:98003
姓名:张三
数学:50
```

外语:60

政治:71

专业一:90

专业二:91^Z (注：^Z表示Ctrl+Z键)

上线考生:

考号	姓名	总分	数学	外语	政治	专业一	专业二
1	丁一	368	70	65	68	80	86
2	王二	358	60	69	62	78	89
4	李四	326	58	50	70	68	80

未上线考生:

考号	姓名	总分	数学	外语	政治	专业一	专业二
3	张三	362	52	60	71	90	91

[例8.12] 在一个m行n列的棋盘上，棋子“马”放在任一指定位置上．写一个程序，输出马连跳三步所有可能的落子位置．

[解题思路]

用二维数组a表示棋盘，马在棋盘中的位置用记录变量p表示．

马的跳步规则如图8.1所示，设当前位置是p，则跳一步可能到达的位置共有8个．可用记录数组常量表示一次跳步后坐标值x,y的增加量．

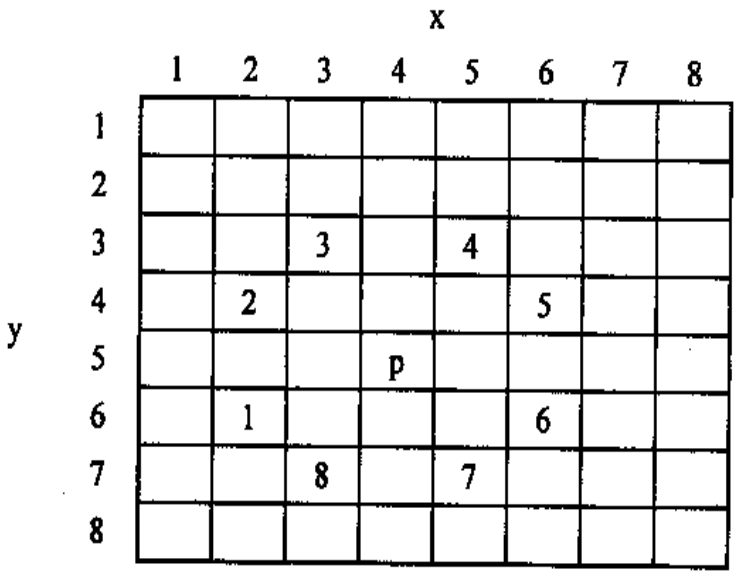


图8.1 马跳一步可能到达的8个位置

init过程用于完成初始化工作：读入起点位置p，用圆点'.'标记该位置，棋盘中其它位置用'0'作标记．

jump(p,k)是个递归过程：从p点出发，连跳k步，把所能到达的位置都

打上星号'*'。

disp过程用于显示棋盘。

[程序]

```
program ch812;
 const m=10;n=9;
       k=3;
 type chessbord=array[1..m,1..n]of char;
      coordinate=record
                   x,y:integer
                 end;
 const d:array[1..8]of coordinate=
          ((x:-2;y:+1),(x:-2;y:-1),
           (x:-1;y:-2),(x:+1;y:-2),
           (x:+2;y:-1),(x:+2;y:+1),
           (x:+1;y:+2),(x:-1;y:+2));
 var a:chessbord;    {表示棋盘}
     p:coordinate;   {表示位置}

 procedure init (var p:coordinate);
 { 初始化 }
  var i,j:integer;
  begin
  for i:=1 to m do
    for j:=1 to n do a[i,j]:='0';
   writeln;
   write('请输入起点坐标:');
   read(p.x,p.y);
   a[p.x,p.y]:='.'; { 做起点标记 }
  end;

 procedure disp;
 { 显示 }
  var i,j:integer;
  begin
   writeln;
```

```
    writeln('可能到达的位置:');
    for i:=1 to m do
    begin
     for j:=1 to n do write(a[i,j]:2);
     writeln
    end
   end;

  procedure jump(p:coordinate;k:integer);
  { 从p点出发跳k步 }
   var s:coordinate;
       i:integer;
   begin
    if k>0 then
     for i:=1 to 8 do  { 依次试探8个方向 }
     begin
      s.x:=p.x+d[i].x;
      s.y:=p.y+d[i].y;
      if (s.x>0)and(s.x<=m)and
        (s.y>0)and(s.y<=n)and { 没有跳出边界 }
        (a[s.x,s.y]='0')      { 没有到过 }
      then begin
             a[s.x,s.y]:='*';  { 做已到标记 }
             jump(s,k-1)       { 跳下一步 }
           end
     end
   end;

  begin
   init(p);
   jump(p,k);
   disp
  end.
```

[运行实例]

请输入起点坐标:5 6

可能到达的位置:

```
0  0  *  0  0  *  *  *  0
0  0  0  *  *  *  0  *  *
0  0  *  0  *  0  *  0  *
0  *  0  *  *  *  0  *  *
*  0  *  *  *     *  *  *
0  *  *  *  0  *  *  *  0
*  0  *  0  *  0  *  0  *
0  *  *  *  0  *  *  *  0
*  0  0  *  *  *  0  0  *
0  *  0  *  0  *  0  *  0
```

[例8.13]　写一个程序：它先从键盘输入一篇英语短文，然后输出一张表格，表中每一行包含两项内容：短文中出现的单词及其出现的次数。

[解题思路]

用记录数组table表示单词表，表中每个元素包含两个域：word域存放单词；count域存放该单词在短文中出现的次数。

定义过程readword(w)，用于识别一个单词，并把它存入w中。

定义过程insert(table,n,w)，用于把w插入到单词表table中，其中，n是表中不同单词的数目。如果表中已有和w相同的单词，则不插入，只将该单词的出现次数加1。

定义过程printtable(table,n)，用于列出单词表。

[程序]

```
program ch813;
 const max=100;                    {假定短文中不同的单词不超过100个}
 type
  element=record
            word:string[20];       {假定单词的最大长度为20}
            count:integer
           end;
  tabletype=array[1..max]of element;
 var w:string;
     n:integer;          { 单词表长度 }
     table:tabletype; { 单词表 }

 procedure readword(var w:string);
 { 读入一个单词w，将其中的字母全换成小写字母 }
  var ch:char;
```

```
        letter:set of char;
    begin
     w:='';
     letter:=['a'..'z','A'..'Z'];
     {找到单词的第一个字母}
     repeat read (ch)
     until eof or (ch in letter);
     {  拼接一个单词  }
     while ch in letter do
     begin
      w:=w+chr(ord(upcase(ch))+32);
      read(ch)
     end
    end;

  procedure insert (var table:tabletype;
                    var n:integer;
                    w:string);
  { 在长度为n的单词表table中查找单词w：若找到了，则令该单词数加
1；若没找到，则将该单词插入表中，并令n加1 }
    var i:integer;
        r:element;
    begin
     r.word:=w;r.count:=0;
     table[n+1]:=r;
     i:=0;
     repeat i:=i+1
     until table[i].word=w;
     table[i].count:=table[i].count+1;
     if i=n+1 then n:=n+1
    end;

  procedure printtable (var table:tabletype;
                        n:integer);
  { 输出单词表 }
```

```
  var i:integer;
  begin
   writeln('统计结果:');
   writeln('单词':20,'      次数');
   for i:=1 to n do
    with table[i] do
    writeln(word:20,count:8)
  end;

 begin
  writeln('请输入短文(用Ctrl-z结束)');
  n:=0;
  while not eof do
  begin
   readword(w);
   if w<>'' then insert(table,n,w)
  end;
  printtable(table,n)
 end.
```

[运行实例]

（略）

在readword过程中，为拼接一个单词使用了标准函数upcase、ord和chr．先用upcase过程将读入的字母一概转换成大写字母upcase(ch)，然后求出它的序号 ord(upcase(ch))．由于大写字母的序号比对应小写字母的序号小32，因此，对应的小写字母是chr(ord(upcase(ch))+32)．

习 题 八

8.1 填空题。

(1) 集合的元素类型可以是_____类型、_____类型或_____类型。

(2) 集合所包含的元素个数由_____决定，最多只能有_____个。

(3) 集合的值是一个_____的元素序列，它与元素的排列顺序_____。

(4) 集合与集合之间可以进行_____、_____、_____、_____、_____等5种运算。

(5) 用于检测某个数据是不是集合元素的运算符是_____。

(6) 给集合变量设置初始值不能用_____语句，只能用_____语句。

(7) 记录和一维数组都是由一组相关的元素构成的，它们的主要区别在于_____。

(8) 引用记录变量中某个域的方法是_____。

(9) 在定义带变体的记录类型时,应先写_____部分，后写_____部分。

(10) 对带变体的记录类型来说，_____域总是可访问的。

8.2 设有集合

a=[1..5]; b=[2,4,6,8,10]; c=[1,3,5];

d=[ ]; e=[2]; f=[1..5,8,15..20]

请计算下列表达式的值：

(1) a=c

(2) b<=f

(3) a+b-c

(4) a*b+b*f

(5) a*d-f

(6) a+f=f

(7) a>=c

(8) 15 in(f-a-b)

8.3 设有如下类型定义和变量说明：

```
type
  coordinate=record  {坐标类型}
    degrees:0..180;  {度}
    minutes:0..60;   {分}
    seconds:0..60;   {秒}
    direction:(north,south,east,west) {方位}
              end;
  location=record    {位置类型}
    latitude:coordinates; {纬度}
    longitude:coordinates; {经度}
            end;
var  city:record
           name:string[10];   {城市名字}
           position:location  {地理位置}
```

```
        end;
```

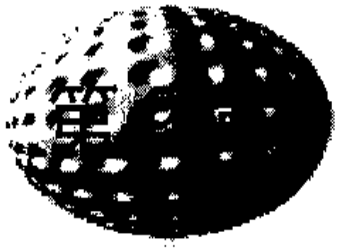
已知名字为abc的城市位于东经125°23′15″,北纬35°21′33″，请对变量city进行赋值。

8.4　写一个按下列规则做游戏的程序:首先,输入整数n(0≤n≤10)；然后，随机产生n个不同的值在0~2n之间的整数(不显示)；接着，再输入n个整数；如果输入的整数和计算机产生的整数有k个相同(不计顺序)，就表示你得了k分。重复若干次后，输出你的平均得分。

8.5　用记录类型

```
element=record
  id:1..50;      {学号}
  math,phys,chem,biol:real; {数学,物理,化学,生物单科分数}
  ave:real       {4科平均分数}
      end
```

描述一个学生的信息,重新编写例6.13所要求的程序。



文 件

程序所要处理的数据，除了通过键盘输入外，还可以从磁盘读取；程序的处理结果，除了显示在屏幕上或打印输出外，还可以写入磁盘。存放在磁盘上的数据都被组织成文件的形式，称为数据文件，简称文件。当数据量较大时，使用数据文件进行输入、输出的优越性尤为明显。因为，数据文件一旦建立，就可以永久保存、重复使用（就像一个Pascal程序文件一旦建立，就可以重复执行一样）。

存放在磁盘上的数据文件是独立于程序的，它们可以在程序执行之前就存在，也可以在程序执行之后才产生。每个数据文件都有自己的名字，称为外部文件名。执行DOS的DIR命令，可以列出这些数据文件名。

如果要使用某个数据文件，则首先要在程序的说明部分说明一个文件变量，这个文件变量称为内部文件名；然后用assign语句使内部文件名和那个要使用的数据文件的外部文件名建立对应关系。计算机系统对于每个经过说明的文件变量都自动附设一个“文件指针”，不管文件中有多少个元素，对文件的读、写操作都只能在文件指针当前所指位置处进行。而且，在进行了一次读、写操作后，文件指针会自动往后移，指出下一个读、写位置。

在程序中使用数据文件的方法，可概括为以下5个步骤：

① 说明文件变量；

② 用assign语句使内部文件名和外部文件名建立对应关系，找到所要处理的数据文件；

③ 用reset语句、rewrite语句“打开”文件，使文件做好读、写准备；

④ 用read语句、write语句对文件进行读、写操作；

⑤ 用close语句“关闭”文件，使内部文件名和外部文件名脱离关系。

TURBO Pascal语言把数据文件分为类型文件、无类型文件和文本文件

3种。无论是哪一种文件，其assign语句的格式都是相同的：

```
assign(内部文件名,外部文a件名)
```

close语句的格式也是相同的：

```
close(内部文件名)
```

而打开文件的语句和读、写数据的语句则因文件种类的不同在用法上有点差异。

下面分别介绍这3种文件的用法。

9.1 类型文件

类型文件是由一些类型相同的元素构成的。

说明类型文件变量的格式为

```
type
  类型名=file of 元素类型;
var
  内部文件名:类型名
```

其中，file 和of是保留字，元素类型可以是除了文件之外的任何类型。

当然，上面的类型定义和变量说明也可以合并：

```
var
  内部文件名:file of 元素类型
```

9.1.1 建立类型文件

为了把数据写入磁盘而打开类型文件，要用rewrite语句。rewrite语句的格式为

```
rewrite(内部文件名)
```

往类型文件中写数据要用write语句。write语句的格式为

```
write(内部文件名,表达式1,表达式2,…,表达式n)
```

其中，每个表达式的类型必须和文件的元素类型相同。这个语句的功能是，

将每个表达式的值依次写入文件，从文件指针当前所指位置开始写，每写一个，文件指针自动后移，指向下一个位置。

[例9.1] 从键盘输入若干张工资卡，建立一个工资文件。假定每张工资卡上只包含职工的编号、姓名和工资3项内容。

[解题思路]

用记录类型描述工资卡，建立记录类型的文件。

规定职工编号num>0，用num=0作为输入结束的条件。

[程序]

```
type
 element=record       {工资卡类型}
  num     :integer;   {编号}
  name    :string[8]; {姓名}
  salary  :real       {工资}
          end;

procedure getdata(var x:element);
{从键盘读数据,返回一张工资卡x}
 begin
  write('编号 :');
  readln(x.num);
  if x.num>0 then
  begin
   write('姓名 :');
   readln(x.name);
   write('工资 :');
   readln(x.salary)
  end {如果x.num=0,则返回一张空卡}
 end;

procedure setfile(fn:string);
{创建工资文件，文件名由参数fn传入}
 var
  f:file of element;
  x:element;
 begin
```

```
  assign(f,fn);
  rewrite(f);
  getdata(x);    {读第一张卡}
  while x.num>0 do {如果读入的不是空卡}
  begin
   write(f,x);   {x写入文件}
   getdata(x)    {读下一张卡}
  end;
  close(f)
 end;

begin
 setfile('GZ1998')
end.
```

运行这个程序后，将在磁盘上建立一个以GZ1998命名的数据文件。

9.1.2　从类型文件中读取数据

为了从磁盘中读数据而打开类型文件要用reset语句．reset语句的格式为

```
reset(内部文件名)
```

从类型文件中读数据要用read语句．read语句的格式为

```
read(内部文件名,变量1,变量2,…,变量n)
```

其中，每个变量的类型必须和文件的元素类型相同．这个语句的功能是，从文件指针当前所指位置开始读，每读一个元素，文件指针自动后移，指向下一个元素．读出的元素依次存入变量1，变量2，…，变量n中．

当文件为空（即文件中没有元素），或文件指针已移到最后一个元素的后面时，继续执行read语句将出现错误．为了避免这种情况的发生，在执行read语句前最好先用标准函数eof测试一下文件的状态：如果刚打开的文件是空文件，则eof函数值为真，否则为假；如果文件指针已移到文件末尾，则eof函数值为真，否则为假．在eof函数值为真的情况下，不能执行read语句．

当标准函数eof用于数据文件（包括类型文件、无类型文件和文本文件）时，其格式为

eof(内部文件名)

[例9.2] 从GZ1998文件中读出全部数据，并显示在屏幕上.

[程序]

```
type
 element=record
   num    :integer;   {编号}
   name   :string[8]; {姓名}
   salary :real       {工资}
           end;

procedure putdata(x:element);
{显示记录x}
 begin
  with x do
    writeln(num:4,name:10,salary:10:2)
 end;

procedure usefile(fn:string);
{从工资文件中读数据，文件名由参数fn传入}
 var
  f:file of element;
  x:element;
 begin
  assign(f,fn);
  reset(f);
  while not eof(f) do {没有遇到文件结束符}
  begin
   read(f,x); {从文件中读一个记录,存入x}
   putdata(x) {在屏幕上显示x}
  end;
  close(f)
 end;

begin
 usefile('GZ1998')
end.
```

9.1.3　扩展类型文件

假设磁盘上已存在一个类型文件，该文件的元素类型是已知的．如果想在这个文件的末尾再添加一些相同类型的新元素，则可采取以下6个步骤：

① 说明文件变量；
② 用assign语句使内部文件名和外部文件名建立对应关系；
③ 用reset语句打开文件；
④ 用seek语句将文件指针移到文件末尾；
⑤ 用write语句写入新增数据；
⑥ 用close语句关闭文件．

步骤④中的seek语句用于在类型文件中移动文件指针，其格式为

```
seek(内部文件名,元素序号)
```

在类型文件中，每个元素都对应一个序号，第1个元素的序号为0，第2个元素的序号为1，依此类推．最后一个元素的序号可用标准函数filesize计算：

```
filesize(内部文件名)
```

[例9.3]　扩展GZ1998文件，在文件的末尾再添加若干记录．

[程序]

```
type
 element=record
   num    :integer;     {编号}
   name   :string[8];   {姓名}
   salary :real         {工资}
           end;

procedure getdata (var x:element);
 begin
  write('编号 :');
  readln(x.num);
  if x.num>0 then
  begin
```

```
    write('姓名 :');
    readln(x.name);
    write('工资 :');
    readln(x.salary)
   end
  end;

 procedure addfile(fn:string);
 {在文件末尾添加新记录，文件名由参数fn传入}
  var
   f:file of element;
   x:element;
  begin
   assign(f,fn);
   reset(f);
   seek(f,filesize(f));{把文件指针移到文件的末尾}
   getdata(x);
   while x.num>0 do
   begin
    write(f,x);
    getdata(x);
   end;
   close(f)
  end;

 begin
  addfile('GZ1998')
 end.
```

利用seek语句还可以实现对类型文件进行随机访问：只要指定元素序号就可以立即找到该元素，并进行读、写操作。

[例9.4] 显示GZ1998文件中第i个元素的值。

[程序]

```
type
 element=record
  num    :integer;  {编号}
```

```
    name   :string[8]; {姓名}
    salary :real       {工资}
           end;
  var i:integer;

  procedure putdata(x:element);
   begin
    with x do
      writeln(num:4,name:10,salary:10:2)
   end;

  procedure recall(fn:string; i:integer);
  {读取第i个元素}
   var
    f:file of element;
    x:element;
   begin
    assign(f,fn);
    reset(f);
    i:=i-1;   {因为第i个元素的序号是i-1}
    if not eof(f) and
       (i>=0)and(i<=filesize(f))
    then begin
     seek(f,i);   {找到第i个元素}
     read(f,x);
     putdata(x)
    end;
    close(f)
  end;

begin
 write('i:'); readln(i);
 recall('GZ1998',i)
end.
```

9.2 无类型文件

磁盘是一种外部存储器，简称外存。对存放在外存上的数据进行存取操作有一个重要的特点：如果N个数据是连续存放的，则一次读取这N个数据所需时间要比分N次、每次读一个数据所需时间少得多；同样，如果要在外存一片连续的存储区域中写入N个数据，则一次写入这N个数据所需时间要比分N次、每次写1个数据所需时间少得多。

针对外存的上述存取特点，为了提高数据在内、外存之间的传输效率，TURBO Pascal语言引入了无类型文件的概念，并提供了两个用于成块读、写的语句。

为了实现成块读、写，需要在内存设置一个缓冲区。内存缓冲区是内存中一片连续的存储区域，在程序中通常用一维数组表示。

对于磁盘中已存在的某个数据文件，不管它是什么类型的，在程序中一概把它当作无类型文件，用blockread语句进行成块读，一次读多个数据，存入内存缓冲区中。如果要往磁盘中存数据，也是先把数据写入内存缓冲区，待内存缓冲区装满后，再用blockwrite语句进行成块写，一次写多个数据。

为了实现成块读、写，需要在程序中说明一个无类型文件变量，然后用assign语句使它与磁盘上的某个文件建立对应关系。

说明无类型文件变量要用保留字file，其格式为

```
var
  内部文件名:file
```

注意：类型文件的读语句

read(f,V1,V2,…,Vn)

和写语句

write(f,E1,E2,…,En)

形式上也是一个语句读、写N个数据，但是，由于它们分别是语句组

read(f,V1);read(f,V2);…;read(f,Vn)

和

write(f,E1);write(f,E2);…;write(f,En)

的缩写，所以，实际上是进行了N次读和N次写。

9.2.1　成块读

为了读一个已存在的数据文件，首先要用reset语句打开这个文件．用于无类型文件的reset语句的格式为

```
reset(f,recsize)
```

其中，f是内部文件名；recsize是整型表达式，其值以字节数为单位，表示元素大小。

如果无类型文件变量f对应于磁盘中的某个类型文件，且该文件的元素类型是已知的，例如，文件的元素类型为element，则一般就取recsize=sizeof(element)．这里，sizeof是个标准函数，用来计算某种类型的数据所占字节数，其格式为

```
sizeof(类型名)
```

用于成块读的blockread语句的常用格式为

```
blockread(f,buf,count,result)
```

其中，f是内部文件名；buf是内存缓冲区变量，通常是一个一维数组；count和result都是整型变量，前者表示一次应该读多少个元素，后者表示一次实际读了多少个元素。如果文件长度（即文件中元素的个数）正好是count的整数倍，则每次执行blockread的结果，result都等于count；如果文件长度不是count的整数倍，则最后一次执行blockread的结果，result将比count小．如果buf是一维数组，则数组的元素个数一般等于count．count取值大小和reset语句中第二个参数recsize的取值有关，要求count与recsize的乘积不大于65535（即64KB）。

[例9.5]　统计GZ1998文件中全体职工的工资总额．

[程序]

```
type
 element=record
   num     :integer;   {编号}
   name    :string[8]; {姓名}
   salary  : real      {工资}
         end;
```

```
procedure getsum(fn:string);
 const
  count=500;
 var
  f:file;
  buf:array[1..count]of element;
  i,result:integer;
  sum:real;
 begin
  assign(f,fn);
  reset(f,sizeof(element));
  sum:=0;
  while not eof(f) do
  begin
   blockread(f,buf,count,result);
   for i:=1 to result do
    sum:=sum+buf[i].salary
  end;
  close(f);
  writeln;
  writeln('工资总额:',sum:10:2)
 end;

begin
 getsum('GZ1998')
end.
```

9.2.2 成块写

为了在磁盘上创建一个数据文件，首先要用rewrite语句打开这个文件，用于无类型文件的rewrite语句的格式为

```
rewrite(f,recsize)
```

其中，f是内部文件名，recsize的含义前面已作详细说明，在此不重复。

用于成块写的blockwrite语句的常用格式为

```
blockwrite(f,buf,count)
```

其中，f是内部文件名；buf是内存缓冲区变量，通常是一个一维数组；count是整型变量，表示一次写多少个元素。在buf是一维数组的情况下，如果写入文件的元素个数正好是数组元素个数的整数倍，则count一般就等于数组元素个数；如果写入文件的元素个数不是数组元素个数的整数倍，则最后一次执行blockwrite时count的值就小于数组元素个数。count取值大小和rewrite语句中第二个参数recsize的取值有关，也是要求count与recsize的乘积不大于65535（即64KB）。

[例9.6]　创建一个新的工资文件GZ1999，与原先使用的GZ1998文件相比，新文件中每个人的工资额增加了10%。

[程序]

```
type
 element=record
  num    :integer;     {编号}
  name   :string[8];   {姓名}
  salary :real         {工资}
          end;

procedure updata(fn1,fn2:string);
 const
  count=500;
 var
  f1,f2:file;
  buf:array[1..count]of element;
  i,result:integer;
 begin
  assign(f1,fn1);
  assign(f2,fn2);
  reset(f1,sizeof(element));
  rewrite(f2,sizeof(element));
  while not eof(f1) do
  begin
   blockread(f1,buf,count,result);
   for i:=1 to result do
```

```
        buf[i].salary:=1.10*buf[i].salary;
      blockwrite(f2,buf,result)
    end;
    close(f1);
    close(f2)
  end;

begin
  updata('GZ1998', 'GZ1999')
end.
```

9.3 文本文件

文本文件是由一些行组成的，每一行中可以有数目不等的元素，各元素的类型也可以不相同，行与行之间用换行符分隔。

说明文本文件变量用预定义类型标识符text，其格式为

```
var 内部文件名:text
```

9.3.1 建立文本文件

在磁盘上建立文本文件有两种途径，一种是用编辑器编辑，另一种是在程序中用write或writeln语句写入。

用编辑器编辑文本文件和用编辑器编辑Pascal源程序文件方法完全相同。其实，源程序文件就是文本文件，由字符行构成。

下面介绍如何由程序创建文本文件。

为了把数据写入磁盘而打开文本文件，要用rewrite语句。rewrite语句的格式为

```
rewrite(内部文件名)
```

往文本文件中写数据可以用write语句，也可以用writeln语句。

write语句的格式为

```
write(内部文件名,输出项1,输出项2,…,输出项n)
```

这个语句的功能是，把每个输出项顺序写入指定文本文件，从文件指针当

前所指位置开始写，每写入一个输出项，文件指针自动后移一个位置，写完最后一个输出项后不换行。

writeln语句有两种格式。

格式1：

```
writeln(内部文件名)
```

格式2：

```
writeln(内部文件名,输出项1,输出项2,…,输出项n)
```

第一种格式writeln语句的功能是，在指定文本文件上执行一次换行操作，即在文件指针当前所指位置自动写入一个不可见的换行符，然后把文件指针移到下一行的开头。

第二种格式writeln语句的功能相当于顺序执行下面两个语句：

```
write(内部文件名,输出项1,输出项2,…,输出项n);
writeln(内部文件名)
```

写入文本文件的每个输出项可以是常量、变量或一般的表达式。类型可以是整型、实型、字符型、布尔型和字符串。每个输出项还可以带输出格式，其作用与第3章介绍的完全相同。

9.3.2　从文本文件中读取数据

为了从磁盘中读取数据而打开文本文件要用reset语句。reset语句的格式为

```
reset(内部文件名)
```

从文本文件中读数据可以用read语句，也可以用readln语句。

read语句的格式为

```
read(内部文件名,变量1,变量2,…,变量n)
```

这个语句的功能是，从文件指针当前所指位置开始读，每读入一个元素，文件指针自动后移，指向下一个元素，读出的元素依次存入变量1，变量2，…，变量n中。

readln语句有两种格式。

格式1：

```
readln(内部文件名)
```

格式2：

```
readln(内部文件名,变量1,变量2,…,变量n)
```

第一种格式readln语句的功能是，在指定文本文件上执行一次换行操作，使文件指针指向下一行的开始位置。

第二种格式readln语句的功能相当于顺序执行下面两个语句：

```
read(内部文件名,变量1,变量2,…,变量n);
readln(内部文件名)
```

read语句和readln语句中的每一个变量的类型只能是整型、实型、字符型或字符串。

注意：写入文本文件的数据除了这4种类型外，还可以是布尔型。但是布尔型数据不能用read或readln直接读取，只能通过条件语句间接赋值得到。具体方法请参考例3.6。

在从文本文件读取数据的应用程序中，经常要用到两个标准函数eoln和eof。eoln用来测试文件指针是否已移到行的末尾：如果文件指针已移到行的末尾，则eoln函数值为真，否则为假。eof用来测试文件指针是否已移到文件的末尾：如果文件指针已移到文件的末尾，则eof函数值为真，否则为假。

eoln函数的格式为

```
eoln(内部文件名)
```

eof函数的格式为

```
eof(内部文件名)
```

9.3.3 应用举例

[例9.7] 从GZ1998文件中读取数据，建立一个以GZ1998.TXT命名的文本文件。

[程序a]

```
type
 element=record
   num    :integer;  {编号}
   name   :string[8]; {姓名}
```

```
    name   :string[8]; {姓名}
    salary :real       {工资}
           end;
var
 f1:file of element;
 f2:text;
 x:element;
begin
 assign(f1,'GZ1998');
 reset(f1);
 assign(f2,'GZ1998.TXT');
 rewrite(f2);
 while not eof(f1) do
 begin
  read(f1,x);
  writeln(f2,x.num:4,x.name:10,x.salary:10:2)
 end;
 close(f1);
 close(f2);
end.
```

在由[程序a]产生的GZ1998.TXT文本文件中，每一行表示一张工资卡片。程序运行结束回到编辑状态后，按功能键F3，输入文件名GZ1998.TXT，可以看到程序的运行结果。

在GZ1998.TXT文件中，每一行包含3种类型数据：整数、字符串、实数。由于字符串不是一行中的最后一个数据，所以，当其它程序从GZ1998.TXT中读取数据时将会遇到麻烦。

为了便于其它程序从GZ1998.TXT中读取数据，可以换一种写入方式，例如每行只写一个数据。

[程序b]

```
type
 element=record
   num    :integer;  {编号}
   name   :string[8]; {姓名}
   salary :real      {工资}
```

```
            end;
var
 f1:file of element;
 f2:text;
 x:element;
begin
 assign(f1,'GZ1998');
 reset(f1);
 assign(f2,'GZ1998.TXT');
 rewrite(f2);
 while not eof(f1) do
 begin
  read(f1,x);
  writeln(f2,x.num);
  writeln(f2,x.name);
  writeln(f2,x.salary:2:2)
 end;
 close(f1);
 close(f2)
end.
```

[例9.8] 从GZ1998.TXT文件中读取数据，以表格形式输出.

[程序]

```
var
 f:text;
 x1:integer;
 x2:string[8];
 x3:real;
begin
 assign(f,'GZ1998.TXT');
 reset(f);
 while not eof(f) do
 begin
  readln(f,x1);
  readln(f,x2);
```

```
    readln(f,x3);
    writeln(x1:4,x2:10,x3:10:2)
  end;
  close(f)
end.
```

[例9.9] 从磁盘中读取一个Pascal源程序文件，将其中的小写字母全部换成大写字母，并换名存储。

[程序]

```
var
  f1,f2:text;
  fn:string;
  c:char;
begin
  write('老文件名:');
  readln(fn);
  assign(f1,fn);
  write('新文件名:');
  readln(fn);
  assign(f2,fn);
  reset(f1);
  rewrite(f2);
  while not eof(f1) do
   if eoln(f1)
   then begin
     readln(f1);
     writeln(f2)
   end
   else begin
     read(f1,c);
     write(f2,upcase(c))
   end;
  close(f1);close(f2)
end.
```

习 题 九

9.1 填空题。

(1) 数据文件名有两种,在程序的说明部分中出现的是它的_____名,可由DOS的DIR命令显示的是它的_____名。

(2) assign语句的作用是_____。

(3) 为了从磁盘中读数据而打开文件，要用_____语句。

(4) 为了往磁盘中写数据而打开文件，要用_____语句。

(5) close语句的作用是_____。

(6) 由file of 说明的文件是_____文件；由file 说明的文件是_____文件；由text说明的文件是_____文件。

(7) 对文件的读、写操作只能在_____位置上进行。

(8) 对无类型文件进行成块读的语句是_____，成块写的语句是_____。

(9) 若f是一个文本文件，则当_____时，eof(f)的值为true，当_____时，eoln(f)的值为true。

(10) 在磁盘上建立文本文件有两种途径，一种是_____，另一种是_____。

9.2 阅读下列程序,指出它们的功能.

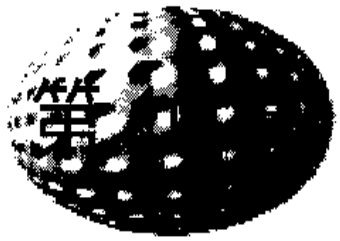
(1)

```
var f:file of integer;
    i,x:integer;
begin
assign(f,'int.dat');
reset(f);
i:=0;
while not eof(f) do
begin
 read(f,x);
 x:=2*x;
 seek(f,i);
 write(f,x);
 i:=i+1
```

```
    end;
    close(f)
  end.
(2)
  type intfile=file of integer;
  var f1,f2,f3:intfile;
  procedure CP(var f,g:intfgile);
   var x:integer;
   begin
    reset(f);
    while not eof(f) do
    begin
     read(f,x);
     write(g,x)
    end;
    close(f)
   end;
  begin
   assign(f3,'int3.dat');
   rewrite(f3);
   assign(f1,'int1.dat');
   CP(f1,f3);
   assign(f2,'int2.dat');
   CP(f2,f3);
   close(f3)
  end.
(3)
  {这个程序在磁盘上的文件名是 PA.PAS}
  type states=(copying,decomment);
  var infile,outfile:text;
      ch:char;
      action:states;
  begin
   action:=copying;
```

```
assign(infile,'PA.PAS');
assign(outfile,'PB.PAS');
reset(infile);
rewrite(outfile);
while not eof(infile) do
 if eoln(infile)
 then begin
  readln(infile);
  writeln(outfile)
 end
 else begin
  read(infile,ch);
  case action of
   copying:
    if ch='{'
    then action:=decomment
    else write(outfile,ch);
   decomment:
    if ch='}'
    then action:=copying
  end
 end;
close(infile);close(outfile)
end.
```

9.3 重新编写例8.11所要求的程序:输入数据从文本文件中读取，处理结果写入文本文件。



指　针

前面已介绍的各种数据类型有一个共同特点：在程序执行期间，这些类型的数据占用内存固定的存储单元，其所占内存单元的数目是在进行变量说明时确定的，在程序执行期间不可增加或减少

注：文件的元素个数虽然是可变的，但它们不是存放在内存，而是存放在外存。对程序中说明的每个文件变量，Pascal系统只为它分配固定数目的、少量的内存单元。

这种占用内存单元数目固定不变的数据称为静态数据。

本章要介绍一种新的数据类型——指针类型，它是专门用来构造动态数据的。

10.1　动态数据和单向链表

10.1.1　动态数据

动态数据是在执行程序的时候才建立起来的，它所占用的内存单元的数目可以根据需要随时增加或减少.

动态数据大多是构造类型的，由若干元素构成．这些元素是在程序执行期间产生的：程序需要用到多少元素，系统就为它分配多少内存单元；当程序不再需要某些元素时，系统可以及时回收这些元素所占用的内存单元.

如果程序要处理一批类型相同的数据元素，但是，在编写程序时又不知道元素的大致数目，在这种情况下最好使用动态数据.

10.1.2 单向链表

单向链表是一种最简单的动态构造型数据。一个长度为n的单向链表由n个类型相同的元素构成，其中，n的值是可变的。如果n=0，则称为空链表。习惯上，把构成单向链表的元素称为结点。

单向链表中的结点一般都是记录类型的。在这种记录中，除了包含一些数值域之外，最重要的是还包含一个指针类型的域（称为指针域），用来存放每个结点的“下一个结点”的地址。

一个长度为3，其中每个结点仅包含一个字符型数值域和一个指针域的单向链表如图10.1(a)所示，而图10.1(b)则表示空链表。

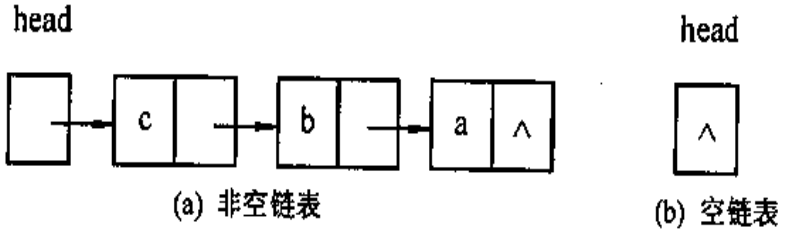


图10.1 单向链表

在图10.1所示的单向链表中，head是指针变量，称为表头指针。对于非空链表，head中存放第一个结点的地址；对于空链表，head中存放“空地址”，在图中用符号“∧”表示。非空链表中的每个结点用一个矩形框表示。矩形框的左边小方格代表数值域，存放一个字符；右边小方格代表指针域，存放下一个结点的地址。由于单向链表是在程序执行期间动态地产生的，因此，这些结点在内存中不一定连续存放（但每个结点所占用的内存区域是连续的）。

对于单向链表，我们不需要知道每个结点在内存中确切的存放位置，只要知道每个结点的下一个结点是谁就可以了。因此，在画单向链表的示意图时，每个结点的指针域中不写下一个结点在内存中具体的存储单元地址，而是引出一条带箭头的线段，表示指向哪个结点。在最后一个结点的指针域中画上符号“∧”，表示后面没有别的结点。

在程序执行期间，单向链表中的结点可以增加，也可以减少：当需要一个结点时，用new语句为它分配内存单元；当不需要某个结点时，用dispose语句释放它所占用的内存单元。因此，使用起来非常方便。

10.2　指针类型和指针变量

10.2.1　指针类型

指针类型定义的格式为

```
指针类型名=^结点类型
```

其中，等号右边跟在箭头后面的结点类型可以是任何类型，但通常是记录类型，用来描述动态构造型数据中的每个元素。

前面曾经提到过，在单向链表中每个结点是一个记录，除数值域之外还包含一个指针域，指向下一个结点——一个相同类型的记录。显然，在定义指针类型时，要用到记录类型，在定义记录类型时，要用到指针类型，无论先定义哪个类型都不符合“标识符必须先定义后使用”的原则。为了解决这个问题，Pascal语言把定义指针类型作为一种特殊情况来对待，允许在定义指针类型时使用一个尚未定义的类型标识符（即箭头后面的结点类型名），先定义指针类型，后定义指针所指向的结点类型。

[例10.1]　定义前面提及的单向链表中的指针类型和记录类型。

```
type
  pt=^node;
  node=record
    data:char;    {数值域}
    next:pt       {指针域}
  end
```

在这个例子中，先用结点类型名node来定义指针类型pt，然后再定义结点类型node。在定义node时又用到了指针类型pt。

10.2.2　指针变量

指针变量用于存放动态数据的内存单元地址。指针变量要用指针类型来说明：

```
var
  指针变量:指针类型
```

在前面各章的实例中，我们经常将类型定义并入变量说明中。但是，对于指针类型来说，类型的定义和变量的说明一般都分开写。

[例10.2] 说明上述单向链表中的表头指针head。

```
type
 pt=^node;      {定义指针类型}
 node=record   {定义结点类型}
  data:char;
  next:pt
       end;
var   head:pt; {说明指针变量}
```

一个指针类型的变量总是占用4个字节的存储空间，用于存放动态数据的内存单元地址。但是，每个指针变量中实际可以放入哪些内容，要根据定义指针类型时所用到的结点类型来决定。本例中，由于结点类型是node类型，因此，指针变量head中可以存放node类型数据的内存单元地址，而不可以存放其它类型数据的内存单元地址。

和其它变量一样，对于一个未经赋值的指针变量来说，它的值也是不确定的，在使用它之前要先进行赋值。关于给指针变量赋值的问题，后面还要详细讨论。

10.2.3 指针常量

保留字nil是各种指针类型的符号常量，表示“空地址”，可以把nil赋值给任何一个指针变量。

在创建和使用动态构造型数据时，要用到常量nil。例如，对于图10.1所示的单向链表，为了表示空链表，可以在代表表头指针head的矩形框中画上符号“∧”，而在程序中则要给变量head赋值nil；为了表示数值为'a'的那个结点是非空链表的最后一个结点，可以在代表该结点指针域的矩形框中画上符号“∧”，而在程序中则要给该结点的指针域赋值nil。

10.3 使用指针变量

存放动态数据所需的内存单元是在运行程序的时候通过执行new语句来分配的。

new语句的格式为

```
new(p)
```

其中，p是一个指针变量。这个语句的功能是，由系统分配一组连续的内存单元给运行中的程序，用于存放动态数据，并将这组内存单元的首地址存入变量p中。

new 语句虽然给动态数据分配了内存单元，但并没有往这些内存单元中存入任何值。那么，如何把数据存入这组内存单元中？如何使用这种动态数据呢？

在Pascal语言中，指针变量指向的结点称为**动态变量**。动态变量在程序中不需要说明，可通过指针变量后跟箭头符号“^”来引用。

假设p是例10.2中定义的pt类型的指针变量。在执行语句new(p)之后，变量p中就有了一个动态变量的地址，这个动态变量在程序中的表示为p^。由于p所指向的结点是node类型的，所以p^是node类型的记录，它的两个域表示为p^.data和p^.next。

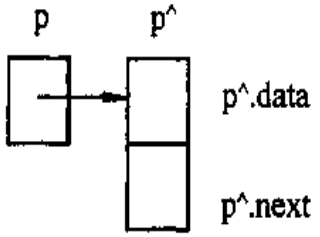
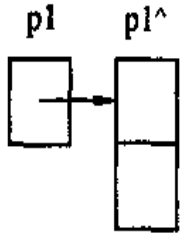


图10.2　new语句执行结果

new(p)语句执行结果可用图10.2表示。

执行new(p)语句，只是产生了一个动态变量p^，并将这个动态变量的地址存入指针变量p中，而这个动态变量的值并没有确定。可以用赋值语句给数值域p^.data和指针域p^.next赋值；如果数值域的类型是整型、实型或字符型，则还可以通过read语句给数值域赋值。



设p1和p2都是前述pt类型的指针变量，执行语句

```
new(p1);new(p2)
```

之后，将产生两个动态变量p1^和p2^，它们的地址将分别存入p1和p2中，如图10.3所示。由于p1^和p2^占用不同的内存单元，因此，条件

```
p1<>p2
```

成立。

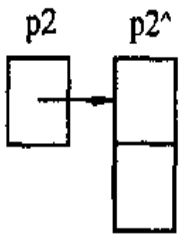


图10.3　产生两个动态变量

此时，如果继续执行语句

```
p1^.data:='a';
p1^.next:=nil;
p2^.data:='b';
p2^.next:=nil
```

则p1^和p2^就有了确定的值，如图10.4所示。

在图10.4所示状态下，执行指针变量间的赋值：

```
p2:=p1;
```

其结果是把p1^的地址复制到p2中，如图10.5所示．由于p1和p2指向同一个动态变量，因此条件

```
p1=p2
```

成立。

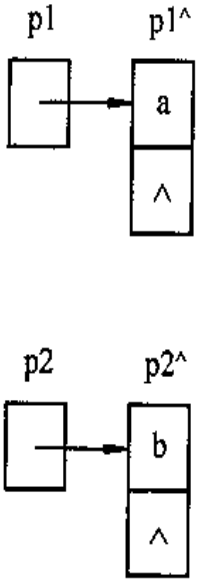


图10.4 给动态变量赋值

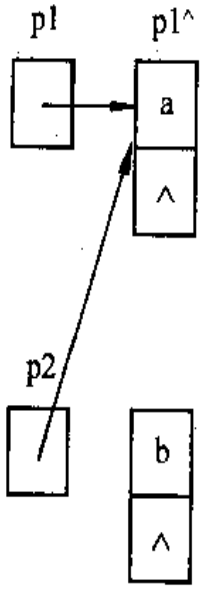


图10.5 p2:=p1

这样，p2中原来保存的那个动态变量的地址就丢失了，那个动态变量仍占用着内存单元，而程序却无法使用它（因为不知道它的地址），造成内存单元的浪费．在程序中，当某个动态变量不再被需要时，可用dispose语句将它所占用的内存单元释放．dispose语句的用法将在10.4.2节中介绍。

在图10.4所示的状态下，执行动态变量间的赋值：

```
p2^:=p1^;
```

其结果是把动态变量p1^整体赋值给p2^，如图10.6所示，相当于执行下面两个赋值语句：

```
p2^.data:=p1^.data;
p2^.next:=p1^.next
```

此时，p1^和p2^这两个动态变量只是值相同，但它们占用不同的内存单元。因此，条件

p1=p2

不成立.

注意：两个指针变量进行相等或不相等的比较、进行赋值操作，都要求这两个指针变量有相同的类型，即指针所指向的结点具有相同的类型。除了这种比较和赋值以外，对指针类型的量（包括变量和常量）不能进行其它任何操作。

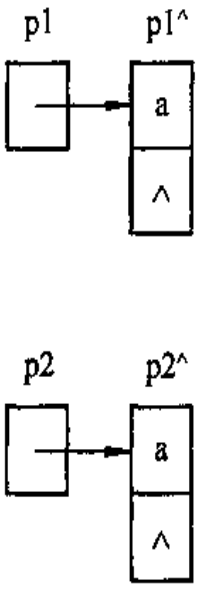


图10.6　p2^:=p1^

[例10.3]　建立一个整型动态变量和实型动态变量.

首先，在程序的说明部分说明指针变量：

```
type
  pt1=^integer;
  pt2=^real;
var
  a:pt1;
  b:pt2;
```

或者，将类型定义并入变量说明中：

```
var
  a:^integer;
  b:^real;
```

然后，在程序的执行部分用new语句创建动态变量：

```
new(a);new(b);
```

执行结果如图10.7所示.

由于a和b是不同类型的指针变量，所以两者之间不能进行比较和赋值．下面的比较运算和赋值操作都是错误的：

```
a=b
a<>b
a:=b
b:=a
```

即便是在执行了赋值语句

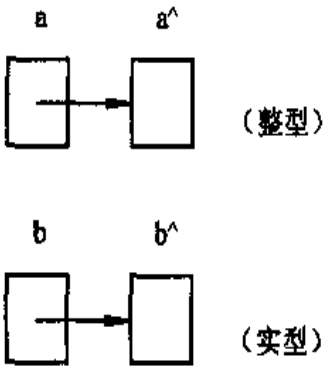


图10.7　产生两个动态变量

```
a:=nil; b:=nil
```

之后，也不允许进行“a=b”或“a<>b”的比较。

两个动态变量之间可以进行哪些运算，取决于这两个动态变量的类型。在例10.3中，由于a^是整型变量，b^是实型变量，所以它们之间可以进行多种算术运算和关系运算，还可以把a^赋值给b^。

10.4 使用单向链表

下面介绍如何建立和使用一个单向链表。所讨论的单向链表以head为表头指针，每个结点有一个指针域和一个字符型数值域，其类型定义为

```
type
 pt=^node;
 node=record
  data:char;
  next;pt
      end;
var
 head:pt { 表头指针 }
```

10.4.1 插入

建立单向链表的过程是一个从无到有不断插入新结点的过程。新结点可以根据需要插入到单向链表的最前面、最后面或者任意两个结点之间。在下面的实例中，每次都是把新结点插到单向链表的最前面，这是一种最简单的插入方法。

单向链表的建立要从空链表开始，建立以head为表头指针的空链表只需要执行一个赋值语句：

```
head:=nil
```

如果一个以head为表头指针的单向链表已存在（可能没有结点，是个空链表），要在最前面插入一个数值域的值为x的新结点，则可用过程insert(x)实现：

```
procedure insert(x:char);
 var p:pt;
```

```
begin
  new(p);          { 给新结点分配内存单元，地址存入p中 }
  p^.data:=x;      { 给新结点的数值域赋值 }
  p^.next:=head;   { 把原来的单向链表接在新结点后面 }
  head:=p          { 修改表头指针，令其指向新结点 }
end;
```

如果想建立图10.1(a)所示单向链表，只需执行下面4个语句：

```
head:=nil;  { 建立空链表 }
insert('a'); { 插入第一个结点 }
insert('b'); { 插入第二个结点 }
insert('c')  { 插入第三个结点 }
```

图10.8描述了这个单向链表的建立过程。

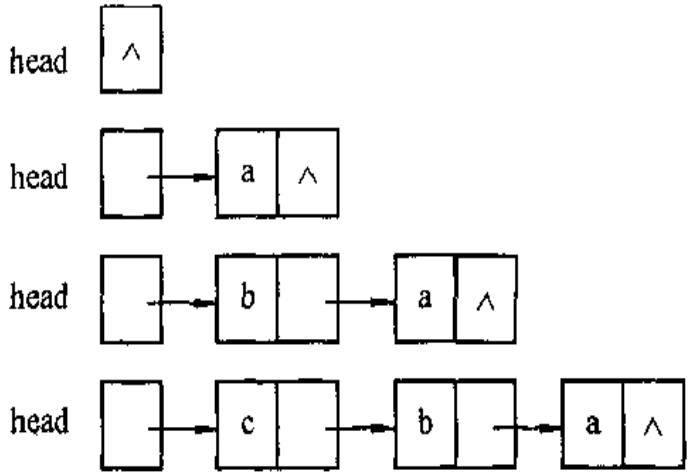


图10.8　建立单向链表

10.4.2　删除

当单向链表中的某个结点不再被需要的时候，可以将它从单向链表中删除，并用dispose语句将它所占用的内存单元释放。

dispose语句的格式为

```
dispose(p)
```

其中，p是指针变量，指向被删除的结点。执行这个语句后，动态变量p^所占内存单元将被释放。不过指针变量p依然存在，因为它是静态变量。

假设要从非空链表中删除由指针变量p所指向的结点p^。可以分两种情

况：

(1) p=head

由于要删除的是第一个结点，所以删除这个结点后，要修改表头指针head，使其指向下一个结点。删除过程如图10.9所示。在程序中用下列语句实现：

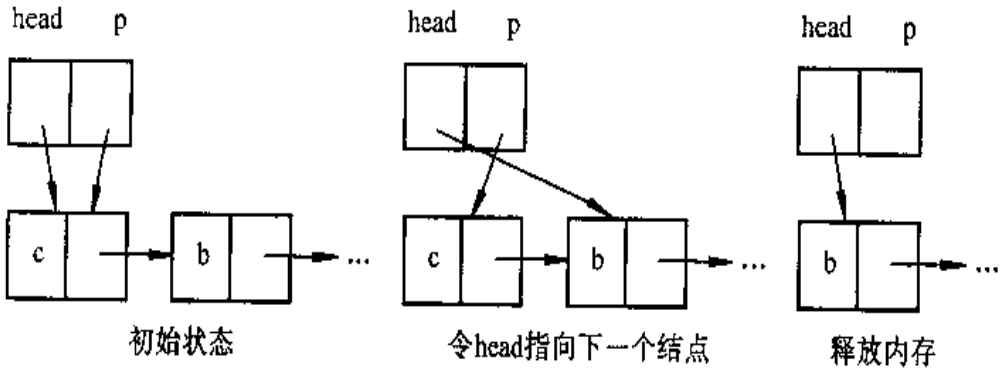


图10.9 删除第一个结点

```
head:=p^.next;
dispose(p)
```

(2) p<>head

由于删除这个结点以后要修改其前面一个结点的指针域的值，所以必须知道待删除结点的前面一个结点的地址。假设这个结点的地址已保存在指针变量q 中，则删除过程如图10.10 所示。在程序中用下列语句实现：

```
q^.next:=p^.next;
dispose(p)
```

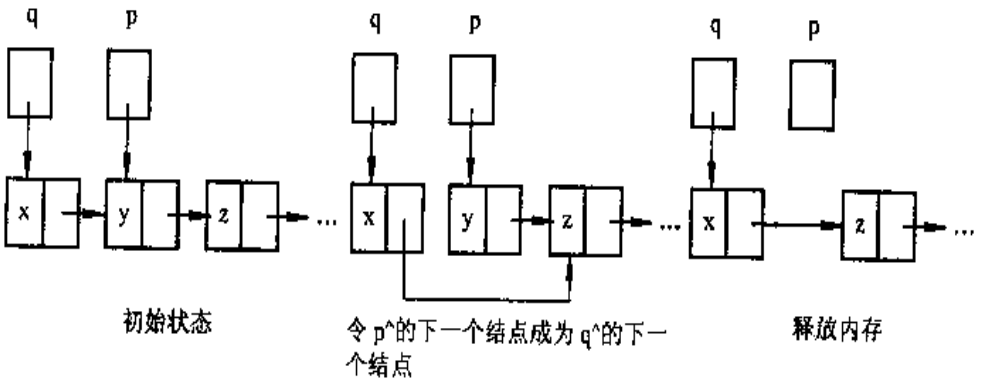


图10.10 删除中间某个结点

10.4.3　遍历

单向链表是动态的构造型数据。若要使用其中的某个元素（即结点），则必须从表头指针开始，顺着保存在每个结点指针域中的指针（即下一个结点的地址）依次往后找。这个过程称为遍历。遍历是单向链表的一种非常重要的操作，对单向链表的许多处理都是建立在遍历的基础上的。

遍历以head为表头指针的单向链表，显示单向链表中每个结点数值域的值，可用过程traverse实现：

```
procedure traverse;
 var p:pt   ;              { p指向当前要访问的结点              }
 begin
  p:=head;                 { 令p指向第一个结点                  }
  while p<>nil do          { 在p还指向某个结点的情况下重复      }
  begin
   write(p^.data);         { 访问p^                             }
   p:=p^.next              { 令p指向下一个结点                  }
  end;
 end
```

10.4.4　应用举例

[例10.4]　某制鞋厂生产学生运动鞋，款式和面料完全相同，大小从32码到39码共有8种规格．已知某批产品经质量检验后已登记入库，质检结果登记在一个以'shoes.dat'命名的Pascal数据文件中，该文件的元素类型描述如下：

```
type
 element=record
  num:integer;
  size:32..39;
  mark:'0'..'3'
       end;
```

其中，num是印在包装盒上的编号，每双鞋有不同的编号；size表示大小规格；mark是质检标记，'3'表示两只鞋都合格，'2'表示左鞋合格，'1'表示右鞋合格，'0'表示两只鞋都不合格．

为提高产品合格率，可对部分不合格产品进行重新包装，例如，假设编号为x的鞋子是左只不合格（右只合格），编号为y的鞋子是右只不合格（左只合格），则可用x的右只和y的左只组合成一双合格品（只要这两双鞋大小规格相同）。

请编写程序，检查数据文件中的每一个元素，并将每双新增合格品是由哪两个包装盒中的鞋组合而成的信息写入一个文本文件中。

[解题思路]

引入一个二维数组A，其行下标用来表示鞋的规格，取值从32到39，列下标为鞋的左右标记，右鞋的取值为'1'，左鞋的取值为'2'；数组元素是一个单向链表的表头指针，链表中结点的数值域为num，记载鞋盒上的编号；所有规格相同、质检标记为'1'的鞋盒上的编号构成一个链表，所有规格相同、质检标记为'2'的鞋盒上的编号构成一个链表。

对从文件中读取的每一个元素进行判别，如果左、右两只鞋都合格或者都不合格，则不作处理，否则，根据size域和mark域的值在相应的链表中增加一个结点，记下num域的值。

令i从32到39，对以A[i,'1']和A[i,'2']为表头指针的链表作同步遍历，并输出各自当前结点中的num域的值。

[程序]

```
program ch104;
 type
  element=record
   num:integer;
   size:32..39;
   mark:'0'..'3';
            end;
  pointer=^node;
  node=record
   num:integer;
   next:pointer
        end;
 var
  a:array[32..39,'1'..'2']of pointer;
  f1:file of element; {输入文件}
  f2:text;                {输出文件}
```

```
  i:integer;
  j:char;
  x:element;
  p,q:pointer;
begin
 {初始化}
 for i:=32 to 39 do
  for j:='1' to '2' do
    a[i,j]:=nil;
 {输入}
 assign(f1,'shoes.dat');
 reset(f1); {打开输入文件}
 while not eof(f1) do
 begin
  read(f1,x);
  if x.mark in['1','2'] {属于一只合格,一只不合格的情况}
  then begin
    new(p);  {为新结点分配存储单元}
    p^.num:=x.num; {记录包装盒上的编号}
    p^.next:=a[x.size,x.mark]; {把新结点插入到对应链表的最前面}
    a[x.size,x.mark]:=p   {修改对应链表的表头指针}
  end
end;
close(f1);
{输出}
assign(f2,'shoes.out');
rewrite(f2); {打开输出文件}
for i:=32 to 39 do
begin {同步遍历两个链表}
  p:=a[i,'2'];    {尺码为i，左只合格的那些鞋的编号构成一个链表，
                   令p指向该链表的第一个结点}
  q:=a[i,'1'];    {尺码为i，右只合格的那些鞋的编号构成一个链表，
                   令q指向该链表的第一个结点}
  while (p<>nil)and(q<>nil) do
```

```
    begin
      writeln(f2,p^.num,'--',q^.num);    {配对鞋的编号写入文本文件}
      p:=p^.next;      {令p指向下一个结点}
      q:=q^.next       {令q指向下一个结点}
    end
  end;
  close(f2)
end.
```

[运行实例]

（略）

习 题 十

10.1 填空题。

(1) 动态数据是在_____的时候建立起来的，大多属于_____类型，由_____构成。

(2) 给动态数据分配存储单元要用_____语句，释放动态数据所占存储单元要用_____语句。

(3) 保留字nil是用来表示_____的。

(4) 指针类型的变量属于_____变量，指针变量指向的结点称为_____变量。

(5) 对指针类型的变量只能进行_____运算和_____操作。

(6) 两个动态变量之间可以进行哪些运算，取决于_____。

(7) 单向链表的长度是指_____。

(8) 对于非空的单向链表来说,表头指针中存放的是_____。

(9) 如果想在程序中使用单向链表，应该先定义_____类型，后定义_____类型。

(10) 从单向链表中删除第一个结点要修改_____的值，删除其它结点要修改_____的值。

10.2 设有类型定义和变量说明如下：

```
type
  pointer=^node;
  node=record
    num:integer;
```

```
    price:real;
    next:pointer
        end;
var
  p1,p2:pointer;
  num:integer;
  price:real;
```

请写出其中的指针类型名、指针变量名、静态变量名、动态变量名。

10.3　对照上题给出的类型定义和变量说明,指出下列表达式和语句中哪些是正确的？哪些是错误的？

(1) p1^num:=1
(2) p2:=p1+1
(3) with p2^. do price:=2.5
(4) p1:=p2
(5) p1^:=p2^
(6) p1^.:=p2^.
(7) read(p1)
(8) read(p1^)
(9) write(p1=p2)
(10) p2^.next:=p1
(11) p2^.next:=num
(12) p2^.next:=p2

10.4　设有类型说明:

```
type
  pointer=^node;
  node=record
    data:integer;
    next:pointer
        end
```

请指出下列子程序的功能:

(1)

```
procedure PA (var f,r:pointer;x:integer);
  var p:pointer;
  begin
    new(p);
    p^.data:=x;
    p^.next:=nil;
    if r=nil
    then f:=p
    else r^.next:=p;
```

```
    r:=p
  end;
```

(2)

```
  procedure PB (var h:pointer);
   var p:pointer;
   begin
    while h<>nil do
    begin
     p:=h;
     h:=h^.next;
     dispose(p)
    end
   end;
```

(3)

```
  procedure PC(h:pointer);
   begin
    if h<>nil
    then PC(h^.next);
    write(h^.data)
   end;
```

10.5 写一个程序:先从键盘输入一系列整数(用Ctrl-z键控制结束),然后按与输入时相反的顺序输出这些整数。

附录 A
TURBO Pascal 编译出错信息选编

错误编号	错误原因
1	内存溢出
2	缺少标识符
3	没有定义标识符
4	标识符重复说明
5	出现非法字符
6	实型常量书写错误
7	整型常量书写错误
8	字符串少右边的单引号
10	源程序文件非正常结束
11	行太长，超过了126个字符
12	类型标识符定义出错
13	打开的文件太多
14	文件名无效或路径不正确
15	在当前目录中找不到文件
16	磁盘满
19	没有定义指针所指向的对象类型
20	缺少变量名
21	类型定义错
22	结构太长
23	集合基类型越界
24	文件的元素类型不可以是文件类型
25	定义字符串类型时，长度值不正确
26	类型不匹配
27	子界基类型不是有序类型
28	定义子界类型时，下界大于上界

29	该处应该是有序类型
30	该处应该是整型常量
31	缺少常量
32	缺少整型或实型常量
33	缺少指针类型标识符
34	函数值类型不正确
36	缺少 begin
37	缺少 end
38	缺少整型表达式
39	缺少有序类型表达式
40	缺少布尔类型表达式
41	运算符和运算对象不匹配
42	表达式不正确
43	非法赋值
44	缺少域标识符
50	缺少 do
54	缺少 of
57	缺少 then
58	缺少 to 或 downto
59	子程序用forward说明，但其后没有定义
60	子程序太多
62	除数为零
63	标准过程或函数不能处理这种类型的文件
64	标准过程或函数不能处理这种类型的变量
65	缺少指针变量
66	缺少字符串变量
67	缺少字符串表达式
74	常量和case类型不匹配
75	缺少记录变量
76	常量越界
77	缺少文件变量
79	缺少整型或实型表达式
85	缺少 ";"
86	缺少 ":"

87	缺少 ","
88	缺少 "("
89	缺少 ")"
90	缺少 "="
91	缺少 ":="
92	缺少 "["
93	缺少 "]"
94	缺少 "."
95	缺少 ".."
96	变量太多
97	for 语句的循环控制变量类型不正确
98	缺少整型变量
101	记录类型常量的域没有按原先定义时的顺序写
102	缺少字符串常量
103	缺少整型或实型变量
104	缺少有序类型的变量
106	缺少字符表达式
113	语句错误
120	缺少 nil
121	无效的限定符
124	语句太长
126	文件类型的参数必须是变量形参
132	严重的磁盘错误
134	表达式错误结束
137	对结构变量进行非法操作
143	调用子程序的方法不正确

附录 B

字符集

字符	序号	字符	序号	字符	序号	字符	序号
空格	32	8	56	P	80	h	104
!	33	9	57	Q	81	i	105
"	34	:	58	R	82	j	106
#	35	;	59	S	83	k	107
$	36	<	60	T	84	l	108
%	37	=	61	U	85	m	109
&	38	>	62	V	86	n	110
'	39	?	63	W	87	o	111
(	40	@	64	X	88	p	112
)	41	A	65	Y	89	q	113
*	42	B	66	Z	90	r	114
+	43	C	67	[	91	s	115
,	44	D	68	\	92	t	116
-	45	E	69	]	93	u	117
.	46	F	70	^	94	v	118
/	47	G	71	-	95	w	119
0	48	H	72	'	96	x	120
1	49	I	73	a	97	y	121
2	50	J	74	b	98	z	122
3	51	K	75	c	99	{	123
4	52	L	76	d	100	|	124
5	53	M	77	e	101	}	125
6	54	N	78	f	102	~	126
7	55	O	79	g	103		127

参考文献

1 Pascal程序设计. 蒋国南译. 北京：清华大学出版社. 1981.

2 谭浩强等编著. Pascal程序设计. 北京：清华大学出版社，1995.

3 王晓敏等编著. Pascal语言程序设计. 北京：人民邮电出版社，1998.